TECHNISCHE HOGESCHOOL TWENTE

AFDELING DER ELEKTROTECHNIEK

Kenmerk:1241.1464

Datum: 13-aug-1976.

Bloem.

TECHNISCHE HOGESCHOOL TWENTE

GEBRUIKSHANDLEIDING

PDP 11 COMPUTERS

REGELTECHNIEK en
AUTOMATISERING,

J.W.MEERMAN

1976

Het is niet VERBODEN ook documentatie van DIGITAL te raadplegen.

INHOUD

HOOFDSTUK I:	**INTRODUCTIE PDP11**	**1.1**
1.0	**Algemeen**	**1.1**
1.1	**Bediening PDP11**	**1.1**
	1.1.1. Adres register	*1.1*
	1.1.2. Data register	*1.1*
	1.1.3. Switch register	*1.3*
	1.1.4. Indicatoren	*1.3*
	1.1.5. Bedieningsschakelaars	*1.3*
	1.1.6. Gebruik bedieningsschakelaars	*1.4*
1.2	**Opbouw van het geheugen**	**1.5**
	1.2.1. Opbouw van geheugenwoorden	*1.5*
	1.2.2. Organisatie van het geheugen	*1.5*
1.3	**Algemene registers**	**1.6**
1.4	**Processor status woord**	**1.7**
1.5	**Instructies**	**1.7**
	1.5.1. Adresseren	*1.7*
	1.5.2. Adresseer modes	*1.8*
	1.5.2.1. Registermode	*1.8*
	1.5.2.2. Register deferred mode	*1.8*
	1.5.2.3. Auto increment mode	*1.8*
	1.5.2.4. Auto increment deferred mode	*1.9*
	1.5.2.5. Auto decrement mode	*1.9*
	1.5.2.6. Auto decrement deferred mode	*1.10*
	1.5.2.7. Index mode	*1.10*
	1.5.2.8. Indexed deferred mode	*1.10*
	1.5.2.9. Specifieke adressering m.b.v. de programma teller	*1.11*
	1.5.2.10 Immediate mode	*1.11*
	1.5.2.11 Absolute mode	*1.11*
	1.5.2.12 Relative mode	*1.11*
	1.5.2.13 Relative deferred mode	*1.12*
	1.5.3. Enkel operand instructies	*1.12*
	1.5.4. Dubbel operand instructies	*1.14*
	1.5.5. Sprong instructies	*1.15*
	1.5.6. Conditie code instructies	*1.16*
	1.5.7. Subroutine instructies	*1.17*
	1.5.8. Operatie instructies	*1.18*
	1.5.9. Floating point instructies (alleen LSI11)	*1.18*
	1.5.10 Interrupties	*1.19*
	1.5.10.1 Interruptie structuur	*1.19*
	1.5.10.2 Non processor request	*1.20*
	1.5.10.3 Interrupt request	*1.20*
	1.5.10.4 Geprogrammeerde interrupts - traps	*1.21*
	1.5.10.5 Hardware fouten - traps	*1.21*
	1.5.10.6 Interruptie instructies	*1.21*

1.6	**Randapparatuur**	1.22
	1.6.1. Teletype / terminal	*1.22*
	1.6.2. Snelle ponsbandlezer / ponser	*1.24*
	1.6.3. AD converter (alleen voor PDP 11/20)	*1.25*
	1.6.4. DA converter / Display Scope (alleen voor PDP 11/20)	*1.26*
	1.6.5. Digitale In- en Uitvoer	*1.29*
	1.6.6. Real time clock (alleen voor PDP 11/20)	*1.30*
HOOFDSTUK 2:	**PROGRAMMERING PDP11 COMPUTERS MET BEHULP VAN DE DEC10**	2.1
2.0	**Inleiding**	2.1
2.1	**fase 1: Het schrijven van een PDP11 programma**	2.1
2.2	**fase 2: Het invoeren resp. wijzigen van het (source) programma**	2.1
2.3	**fase 3: Het vertalen van het PDP11 programma**	2.2
2.4	**fase 4: Het linken of samenknopen**	2.2
2.5	**fase 5: De executie van een PDP11 programma**	2.3
HOOFDSTUK 3	**ASSEMBLER PROGRAMMERING PDP11**	3.1
3.0	**Inleiding**	3.1
3.1	**Statements - Instructie regels**	3.1
	3.1.1. Gegevens label	*3.1*
	3.1.2. Gegevens operator	*3.1.*
	3.1.3. Gegevens Operanden	*3.2*
	3.1.4. Gegevens comment	*3.2*
	3.1.5. Overzicht in de listing	*3.2*
3.2	**Symbolen**	3.2
3.3	**Assignment statements**	3.3
	3.3.1. Algemene register symbolen	*3.3*
3.4	**Expressies (Rekenkundige uitdrukkingen)**	3.4
	3.4.1 Getallen	*3.4*
	3.4.2. Rekenkundige en logische operatoren	*3.4*
	3.4.3. ASCII karakters	*3.4*
	3.4.4. Assembler locatie teller	*3.5*
3.5	**Adressering**	3.5
	3.5.1. Adresseer modes	*3.6*
	3.5.1.1. Register mode	*3.6*
	3.5.1.2. Deferred register mode	*3.6*
	3.5.1.3. Auto increment mode	*3.6.*

3.5.1.4. Deferred autoincrement mode 3.7
3.5.1.5. Auto decrement mode 3.7
3.5.1.6. Deferred auto decrement mode 3.7
3.5.1.7. Index mode 3.7
3.5.1.8. Indexed deferred mode 3.7
3.5.1.9. Direkte operand 3.8
3.5.1.10. Absoluut adres 3.8
3.5.1.11. Relatief adres 3.8
3.5.1.12. Deferred relatief adres 3.8

3.6 Instrukties 3.9

3.7 Assembler besturingscodes 3.9
3.7.1. .TITLE 3.9
3.7.2. .GLOBL 3.9
3.7.3. Programma sectie codes 3.9
3.7.4. Assembler besturingscodes voor het plaatsen van data 3.10
3.7.4.1. .BYTE 3.10
3.7.4.2. .WORD 3.10
3.7.4.3. .ASCII 3.10
3.7.4.4. .ASCIZ 3.11
3.7.4.5. . RAD 50 3.11
3.7.5. Assembler locatie teller besturing 3.11
3.7.5.1. .EVEN 3.11
3.7.5.2. . ODD 3.12
3.7.5.3. .BLKB en .BLKW 3.12
3.7.6. Direkte invoer van floating point getallen 3.12
3.7.7. Scheidingsbesturingscodes 3.13
3.7.7.1. .END 3.13
3.7.7.2. .EOT 3.13
3.7.7.3. .LIMIT 3.13
3.7.8. Conditionele assemblages 3.13

3.8 Macro's 3.14
3.8.0. Inleiding 3.14
3.8.1. Macro definities 3.14
3.8.2. Macro aanroep 3.15

HOOFDSTUK 4: STANDAARD SUBROUTINE PAKKET 4.1

4.0 Inleiding 4.1

4.1 Gebruik standaard subroutines 4.2

4.2 Het samenvoegen van standaard subroutines met het gebruikersprogramma 4.5

4.3 Gebruik foutvector 4.7

4.4 Overzicht en beschrijving van aanwezige subroutines 4.8
4.4.1. Tekst routines 4.9
4.4.2. Conversie routines 4.12

4.4.3. Integer routines 4.14
4.4.4. Floating point routines 4.15
4.4.5. Hulproutines 4.19
4.4.6. Display routines 4.20
4.4.7. Plot routines 4.21

HOOFDSTUK 5: INTRODUKTIE VAN HET RT11 OPERATING SYSTEEM 5.1

5.0 Inleiding 5.1

5.1 RT11 monitor 5.1
5.1.1. Algemene file specificatie 5.2
5.1.2. Starten RT11 5.3
5.1.3. Monitor commando's 5.3

5.2 Systeemprogramma's 5.5
5.2.1. Assembler vertaalprogramma's: MACRO, EXPAND, ASSEMBLER 5.5
5.2.2. Hogere taal vertalers: FORTRAN en BASIC 5.6
5.2.3. Samenknoopprogramma's: LINK en LIBR 5.6
5.2.4. Hulpprogramma: ODT 5.9
5.2.5. Tekst manipulatie programma: EDIT 5.10
5.2.6. File manipulatie programma: PIP 5.11

APPENDIX A: *Computer configuraties van de vakgroep Regeltechniek en Automatisering* A1–A9

APPENDIX B: *PDP11 instructie repertoire* B1–B2

APPENDIX C: *FORTRAN op de PDP11* C1–C10

APPENDIX D: *BASIC op de PDP11* D1–D6

HOOFDSTUK I: INTRODUCTIE PDP11

1.0. Algemeen

De PDP11 serie computers bestaat uit een aantal typen mini- en microcomputers met vrijwel identieke eigenschappen voor algemene toepassingen. Aanwezig bij de Vakgroep Regeltechniek en Automatisering zijn twee minicomputers (PDP 11/20 en PDP 11E10) en een aantal microcomputers (LSI11). Een aantal van de belangrijkste eigenschappen worden hieronder opgesomd:

. Woordlengte 16 bit (2 bytes)
. paralelle logica
. 2-complement arithmetic
. adresseercapaciteit 32768 woorden
  adresseercapaciteit 65536 bytes
. 8 algemene registers
. unibus I/O structuur (m.u.v. LSI 11) (deze heeft een zgn Q bus)
. enkel en dubbel operand instructies
. woord en byte adressering
. eenvoudige adressering randapparatuur registers.

Op een aantal van deze eigenschappen wordt hierna nader ingegaan. Allereerst wordt de bediening van de PDP11 behandeld. Daarna worden enkele eigenschappen nader behandeld en wordt het instructiepakket beschreven. In Appendix A zijn overzichten gegeven van de aanwezige PDP11 computers.

1.1. Bediening PDP11

Op de voorkant van de PDP11 computers zijn een aantal van de hierna te noemen schakelaars en indicators aanwezig.

1: 18 of 16 bits adresregister (indicators)
2: 16 bits data register (indicators)
3: 18 of 16 bits schakelaar register (switch-register)
4: een aantal indicatoren met diverse functies
5: een aantal bedieningsschakelaars

1.1.1. Adresregister

Het 18 of 16 bits adresregister bestaande uit indicators is verbonden met de uitgang van het Bus Address Register. Het register bevat het adres van de data die verwerkt worden. Bij de PDP 11E10 is het adresregister en het dataregister gecombineerd (zie EXAM schakelaar).

1.1.2. Dataregister

Het 18 of 16 bits dataregister bestaande uit indicators bevat de informatie die op dat moment verwerkt wordt. Het dataregister bevat de informatie van de positie aangegeven door het adresregister. Bij de PDP 11E10 is het dataregister met het adresregister gecombineerd.

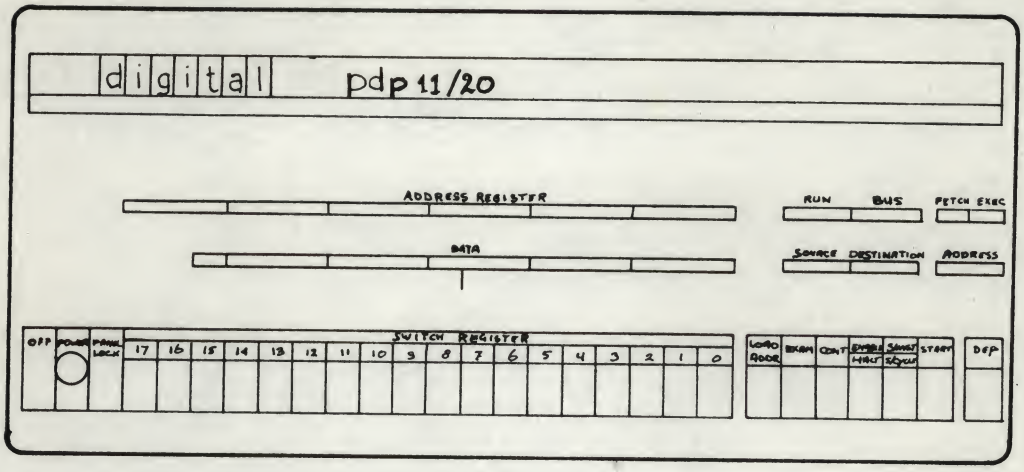


Fig. 1. Schets van de console van de PDP 11/20.

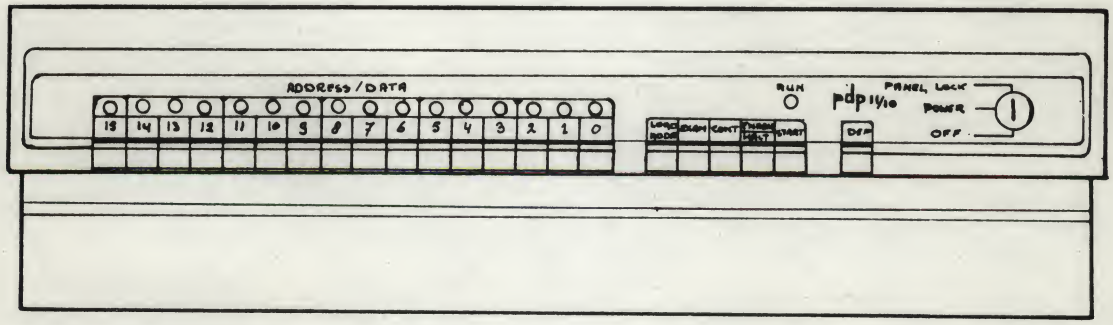


Fig. 2. Schets van de console van de PDP 11/10.

1.1.3. Switchregister

Het switchregister bestaat uit 16 of 18 schakelaars die genummerd zijn van 0 t/m 15 resp. 0 t/m 17. Een schakelaar die omhoog staat correspondeert met een 1. De informatie in het schakelaarregister kan in het adresregister worden overgenomen, of met behulp van het adresregister in elke geheugenplaats (of randapparaatregister) geplaatst worden.

1.1.4. Indicatoren

Afhankelijk van het computertype zijn een aantal van de hierna te noemen indicatoren aanwezig.

RUN	– geeft aan dat de centrale verwerkingseenheid loopt.
FETCH	– geeft aan dat de computer bezig is een instructieregel uit het geheugen te halen.
EXECUTE	– geeft aan dat de centrale verwerkingseenheid bezig is een instructie uit te voeren.
BUS	– geeft aan dat een randapparaat de unibus bestuurt.
SOURCE	– geeft aan dat de centrale verwerkingseenheid met de source data (dubbel operand instructie) bezig is.
DESTINATION	– geeft aan dat de centrale verwerkingseenheid met de destination data bezig is.
ADDRESS	– geeft de source of destination adres cyclus aan. Deze adres cyclus ontstaat alleen als de source of destination data uit het geheugen moet worden gehaald.

1.1.5. Bedieningsschakelaars

OFF–POWER–PANEL LOCK – Sleutelschakelaar

deze schakelaar dient om de netspanning voor de computer in of uit te schakelen. In de stand PANEL LOCK worden alle schakelaars op het frontpaneel geblokkeerd, zodat het dan niet mogelijk is de voortgang van een werkend programma te verstoren.

LOAD ADDR

de informatie uit het schakelaar register (switch register) wordt overgenomen in het adresregister.

EXAM

de informatie op het adres aangewezen door het adresregister verschijnt in het data register. Bij de PDP 11E10 wordt zolang de EXAM schakelaar ingedrukt gehouden wordt het adres (adres register) getoond, terwijl na loslaten van de EXAM schakelaar de inhoud (dataregister) verschijnt.

DEP

de informatie in het schakelaarregister wordt weggezet op het adres aangewezen door het adresregister.

ENABLE/HALT

de stand van deze schakelaar bepaalt of een programmeur kan werken of geblokkeerd wordt. In de stand HALT heeft de schakelaar START geen effekt.

START

start de executie van een programma mits de schakelaar ENABLE/HALT in de stand ENABLE staat. Bij het indrukken van de schakelaar START wordt het systeem geinitialiseerd. Bij het loslaten van de schakelaar

start de executie vanaf de locatie welke het laatst geladen is met de LOAD ADDR schakelaar.

CONT

de programma executie wordt voortgezet vanaf de locatie volgend op de laatst geëxecuteerde locatie. Het systeem wordt niet geïnintialiseerd.

S/INS–S/CYCLE

bepaalt of de centrale verwerkingseenheid 1 instructie ofwel 1 cyclus (instructieonderdeel) uitvoert wanneer de computer in de HALT mode is en de schakelaar CONT bediend wordt.

OPMERKING:

Wanneer een programma geëxecuteerd wordt zijn de schakelaars LOAD ADDR, EXAM en DEP geblokkeerd, zodat verstoring van de executie wordt voorkomen.
Wanneer de programma executie gestopt moet worden, moet de ENABLE/HALT schakelaar in de HALT stand geplaatst worden. De programma executie stopt nu aan het eind van de onderhavige instructie of aan het eind van de onderhavige cyclus afhankelijk van de stand van de schakelaar S/INS-S/CYCLE.

1.1.6. Gebruik bedieningsschakelaars

– Controle inhoud geheugen locatie of randapparaatregister

Plaats HALT/ENABLE schakelaar in de stand HALT. Plaats het adres van de te controleren locatie in het schakelaar register. Laad dit adres met behulp van de LOAD ADDR schakelaar in het adresregister.
Door vervolgens de EXAM schakelaar te gebruiken verschijnt de gewenste informatie in het dataregister.
Wordt daarna opnieuw de EXAM schakelaar gebruikt, dan wordt het adres register met twee verhoogd en verschijnt de inhoud van de volgende locatie in het data register.

– Plaatsen van informatie in een geheugen locatie (of randapparaat register)

Plaats HALT/ENABLE schakelaar in de stand HALT. Plaats het adres van de locatie in het schakelaarregister en druk vervolgens LOAD ADDR.
Plaats de gewenste informatie in het schakelaar register en druk vervolgens DEP.
De informatie wordt nu in de eerder aangegeven locatie geplaatst. Wordt vervolgens EXAM gebruikt, dan zal de zo juist geplaatste informatie in het data register zichtbaar worden.

– Controle en plaatsen van informatie in de Algemene Registers

Controle en het plaatsen van informatie in Algemene Registers vindt op dezelfde manier plaats als hiervoor voor geheugenlocaties aangegeven is. De absolute adressen van de Algemene Registers zijn:

R_0 t/m R_7 : 177700 t/m 177707.

– Starten van een programma

Om een in het geheugen aanwezig programma te starten moet het start adres in het schakelaarregister geplaatst worden. Vervolgens moet de LOAD ADDR schakelaar gebruikt worden, waarmee het startadres in het adresregister geladen is. De schakelaar HALT/ENABLE moet in de stand ENABLE geplaatst worden. Vervolgens moet de START schakelaar ingedrukt worden. Na het loslaten van de START schakelaar is de programma executie gestart.

– *Stap voor stap executie van een programma*

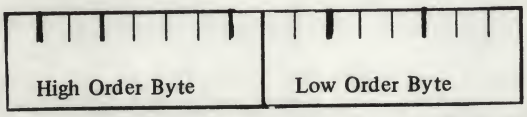
In de test fase kan het voordeel hebben een programmadeel stap voor stap door te lopen. De HALT/ENABLE schakelaar moet daarvoor in de stand HALT geplaatst worden. Afhankelijk van de stand van de S/INS-S/CYCLE schakelaar wordt nu telkens na het drukken van de CONT schakelaar één instructie of één cyclus uitgevoerd. Aan de hand van de indicaties in het adres - en het data register, kan bepaald worden of de programma voortgang correct is.

1.2. Opbouw van het geheugen

De PDP11 heeft een adresseercapaciteit van 32768 woorden of 65536 bytes. Dit betekent dat maximaal, zonder extra voorzieningen, 32 k geheugen aan de PDP 11 zouden kunnen worden geïnstalleerd. Daar echter ook de registers van randapparaten geadresseerd moeten kunnen worden, is de geheugencapaciteit beperkt tot maximaal 28 k. In het gebied van 28 k tot 32 k bevinden zich de adressen van de registers van randapparaten.

1.2.1. Opbouw van geheugen woorden

Het geheugen is opgebouwd uit 16 bits woorden, maar kent een onderverdeling in bytes. Het is mogelijk zowel woordlocaties als bytelocaties te adresseren. In fig. 3 is een voorbeeld gegeven van een geheugenwoord op locatie 1000. Dit woord is opgebouwd uit de bytes op de locaties 1000 en 1001. Het eerstvolgende geheugenwoord bevindt zich op locatie 1002.



Byte adres: 1001 Byte resp. Woordadres 1000

Figuur 3: Opbouw geheugenwoorden.

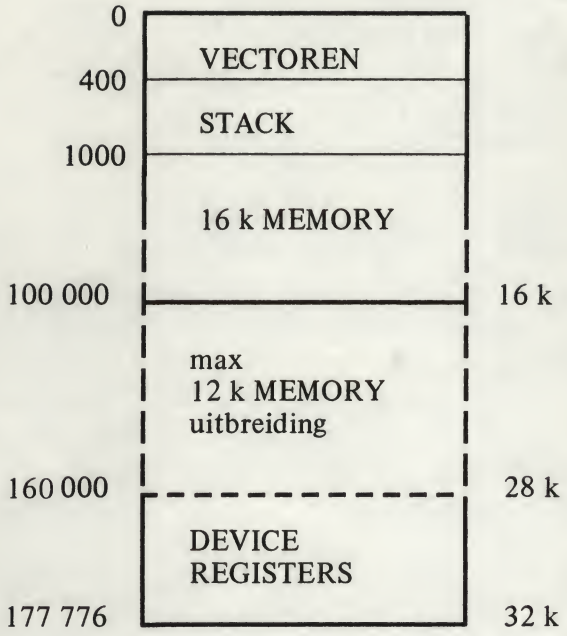
Enkele gevolgen van de byte nummering van het geheugen zijn:
- Woorden moeten altijd even adressen bezitten
- De programmateller kan, omdat een instructie steeds uit één of meer woorden bestaat, gedurende de executie van een programma alleen even getallen bevatten
- In alle instructies die met geheugenverkeer te maken hebben, dient vermeld te worden of de instructie op een woord of een byte betrekking heeft.

1.2.2. Organisatie van het geheugen

Gezien de woordlengte van 16 bits (ook voor de adressering), zal het duidelijk zijn dat er maximaal 64 k verschillende bytes of 32 k verschillende woorden geadresseerd kunnen worden.

In figuur 4 is de geheugen indeling getekend. Hierin zijn de volgende gebieden te onderscheiden:

Adres (octaal):			
0 t/m	400	VECTOREN	In dit gebied bevinden zich de interruptie vectoren., d.w.z. geheugen plaatsen waarin aangegeven is waar het programma bij een interruptie vervolgd moet worden.



Figuur 4 – Opbouw van het geheugen.

400 t/m	1 000	STACK	Aanbevolen als stackruimte (STAPEL-LIFO = Last In First Out data ruimte) voor gebruikers om hardware gegenereerde gegevens zoals het terugkeeradres bij een subroutine aanroep etc. op te slaan
1 000 t/m	77 776	16 k MEMORY	vrij geheugen, waarin gebruikers programma's, data en eventueel het operating system geplaatst kunnen worden
100 000 t/m	157 776	12 k MEMORY	uitbreidingsmogelijkheid van het geheugen tot 28 k
160 000 t/m	177 776	DEVICE REGISTERS	In dit gebied bevinden zich de registers van randapparaten, algemene registers van de processor en processor status woord etc.

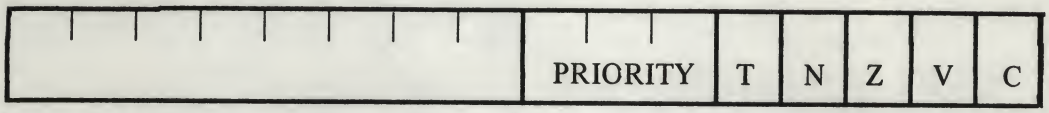
1.3. Algemene registers

De central processor van de PDP 11 is uitgerust met 8 algemene registers (R_0, ...R_7) gelocaliseerd in het geheugengebied boven 28 k woorden (adressen 177 700, ... 177 707). Deze registers kunnen gebruikt worden als accumulator, maar vervullen ook een belangrijke rol bij de berekening van operanden, adressen waarop operaties plaatsvinden. Door middel van een aantal adresseermodes zijn de registers op vele manieren te gebruiken (zie adresseermodes).

Afgezien van deze algemene eigenschappen zijn er aan twee registers speciale functies toegekend. Zo fungeert register 7 als *programma teller* (PC = program counter) en register 6 als *hardware stack wijzer* (SP = stack pointer). Ondanks deze speciale functies zijn deze registers ook algemeen te gebruiken.

1.4. Processor Status register

De central processor bevat, evenals alle randapparaten, een status register. In fig. 5 is een schets gegeven. Dit register bevat informatie over:
– de prioriteit van de central processor
– het resultaat van de laatst uitgevoerd instructie.
De prioriteit van de central processor kan met behulp van een programma instructie worden gezet op verschillende niveaus.



figuur 5 – Processor Status Woord (PSW)

De informatie over de prioriteit bevindt zich in de bitposities 5 t/m 7 van het processor status register. Vier bits van het status register geven informatie over het resultaat van de laatst uitgevoerde instructie die betrekking had op operanden. Sprongen,subroutine aanroepen etc. hebben geen invloed op de status bits.
De status bits worden als volgt geset:

bit 0:	C	als bij de laatste instructie een carry van het meest significante bit opgetreden is.
bit 1:	V	als een rekenkundige overflow heeft plaatsgevonden.
bit 2:	Z	als het resultaat nul was.
bit 3:	N	als het resultaat negatief was.

Het T-bit wordt gebruikt tijdens programma-debugging. Indien dit bit geset wordt,wordt er een interne interrupt (trap) veroorzaakt.
Het adres van het processor status register is 177 776.

1.5. Instructies

De PDP11 kent een aantal typen instructies, zoals b.v.:
– enkel operand instructies,
– dubbel operand instructies,
– sprong instructies.
– conditie code instructies
– subroutine instructies
– interrupt instructies
– floating point instructies
– operatie instructies

In machinecode bestaat een instructieregel uit een aantal velden. Afhankelijk van het type instructie kunnen de volgende velden voorkomen:

1: Instructie code veld
2: Eén of twee adresvelden
3: Sprong offset veld
4: Enkele specifieke velden, zoals codes bij Traps.

Alvorens de verschillende typen instructies behandeld worden, wordt eerst nader ingegaan op de adresseermogelijkheden.

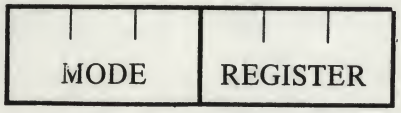
1.5.1. Adresseren

Elke instructie, die met operanden te maken heeft, kent een of twee adresvelden. Elk adresveld bestaat uit 6 bits, onderverdeeld zoals in figuur 6 te zien is in

een 3 bits mode veld en een 3 bits register veld. Elke adressering maakt dan ook van één van de algemene registers gebruik, hetzij direkt, hetzij indirekt met behulp van de programma teller.

In het 3 bits mode veld kunnen 8 verschillende adresseermodes aangegeven worden. In de volgende paragraaf zullen de adresseermodes nader beschouwd worden.

In het 3 bits registerveld kunnen alle 8 algemene registers aangegeven worden.



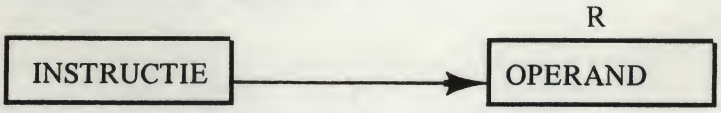
figuur 6: Adresveld van een PDP11 instructie.

1.5.2. Adresseermodes

Zoals in fig. 6 te zien zijn 3 bits aanwezig in het adresveld om de adresseermode aan te geven. Er ontstaan zo dus acht adresseermodes die na elkaar behandeld worden. Daarnaast ontstaan, wanneer de programmateller als register gebruikt wordt een aantal specifieke adresseringsmogelijkheden.

1.5.2.1. Registermode

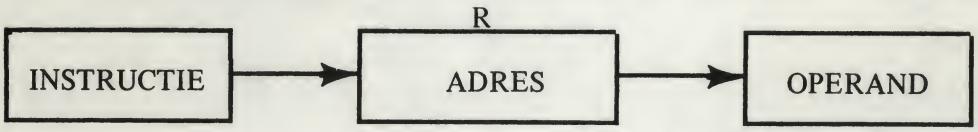
Adresseermode 0 heet de *registermode.* Hiermee is het mogelijk de acht algemene registers te adresseren.



Voorbeeld: CLR R1
Betekenis: R1 wordt nul gemaakt.

1.5.2.2. Register deferred mode

Adresseermode 1 heet de *register deferred mode.* Bij deze mode wordt de inhoud van het register uit het registerveld gebruikt als adres van de operand.

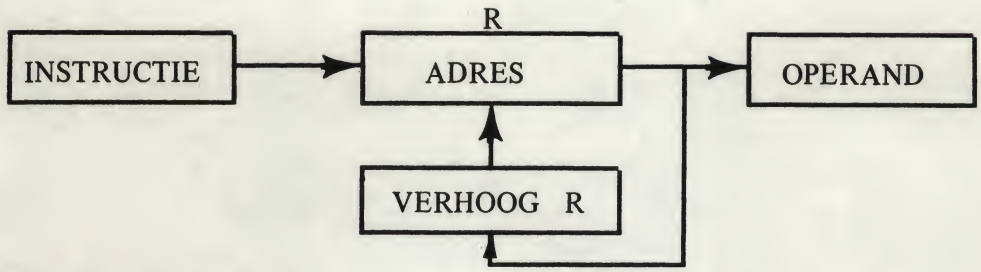


Voorbeeld: CLR (Ro)

Betekenis: De locatie, waarvan het adres in Ro staat wordt nul gemaakt.

1.5.2.3. Auto increment mode

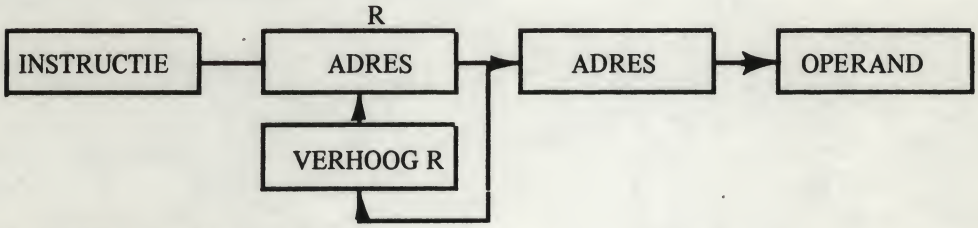
Adresseermode 2 heet de *auto increment mode.* Bij deze adresseermode wordt, evenals bij mode 1, de inhoud van het register uit het regiserveld gebruikt als adres van de operand. Hierna wordt echter het register, met een of twee opgehoogd, afhankelijk van het feit of er sprake is van een byte of woord instructie.



Voorbeeld: CLR (R1) +
Betekenis: De locatie, waarvan het adres in R1 staat, wordt nul gemaakt. Hierna wordt R1, omdat het een woordinstructie betreft, met twee opgehoogd.

1.5.2.4. Auto increment deferred mode

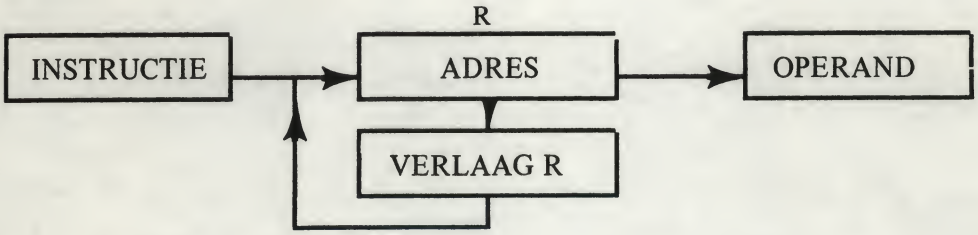
Adresseermode 3 heet: *auto increment deferred mode.* Bij deze adresseermode wordt de inhoud van het register uit het registerveld gebruikt om de locatie aan te wijzen waar het adres van de operand gevonden wordt. Hierna wordt het register opgehoogd, met een als het een byte instructie betreft, en met twee als het een woord instructie betreft.



Voorbeeld: CLR @ (Ro) +
Betekenis: De locatie, aangeduid door de locatie waar Ro naar wijst, wordt nul gemaakt. Hierna wordt Ro met twee opgehoogd omdat het een woord instructie betreft.

1.5.2.5. Auto decrement mode

Adresseermode 4 heet: *auto decrement mode.* Bij deze mode wordt, *alvorens* het register uit het registerveld gebruikt wordt als adres van de operand, dit register met een resp. twee verlaagd, indien het een byte- resp. woordinstructie betreft.

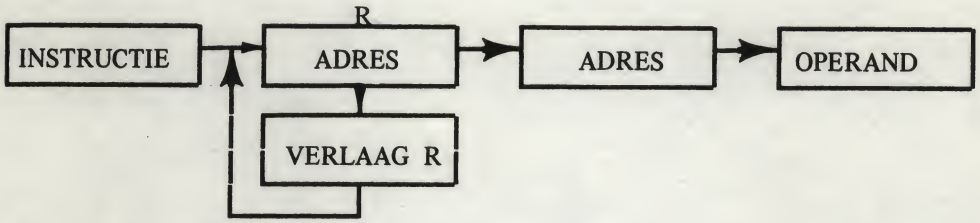


Voorbeeld: CLR –(R4)

Betekenis: Eerst wordt R4, omdat het een woordinstructie betreft, met twee verlaagd. Daarna wordt R4 gebruikt als aanwijzer van de operand.

1.5.2.6. Auto decrement deferred mode

Adresseermode 5 heet: *auto decrement deferred mode.* Bij deze mode wordt, *alvorens* het register uit het registerveld gebruikt wordt, *als aanwijzer van het adres van de operand,* dit register met een resp. twee verlaagd, indien het een byte resp. een woord instructie betreft.

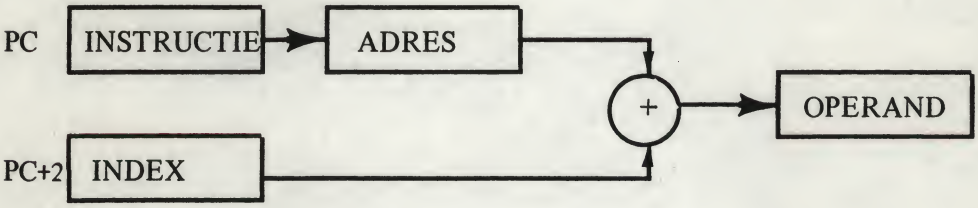


Voorbeeld: CLR -@ (R2)

Betekenis: Eerst wordt R2 - omdat het een woordinstructie betreft - met twee verlaagd. Daarna wordt R2 gebruikt als aanwijzer van het adres van de operand.

1.5.2.7. Index mode

Adresseermode 6 heet: *index mode.* Bij deze mode wordt een index opgeteld bij het register uit het registerveld. De som vormt het adres van de operand. De index wordt geplaatst op de locatie volgend op de instructie. Er is voor deze adresseermode een extra geheugenwoord nodig.

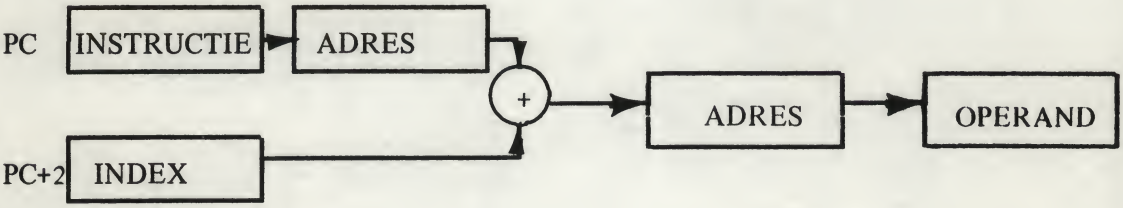


Voorbeeld: CLR X(R2)

Betekenis: De locatie, bepaald door de som van X en de inhoud van R2, wordt nul gemaakt.

1.5.2.8. Indexed deferred mode

Adresseermode 7 heet: *Indexed deferred mode.* Bij deze adresseermode wordt een index opgeteld bij de inhoud van het register uit het registerveld. De som wijst naar de locatie, waar het adres van de operand is te vinden.



Voorbeeld: CLR @X(Ro)

Betekenis: De som van de index X en de inhoud van het register Ro wordt gebruikt om het adres aan te wijzen van de locatie die nul gemaakt wordt.

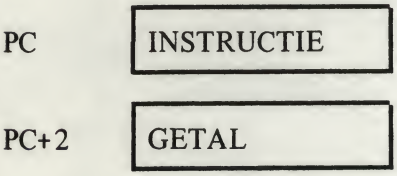
1.5.2.9. Specifieke adressering met behulp van de programmateller

Wanneer als register in het registerveld R7 (de programmateller) is gebruikt, ontstaan een aantal specifieke adresseermogelijkheden. Binnen de assembleer taal PAL11 resp. MACRO 11 worden deze mogelijkheden benut. In figuur 7 zijn de mogelijkheden in een tabel geplaatst.

MODE	REGISTER	BESCHRIJVING	PROGRAMMERING
2	7	immediate	#A
3	7	absolute	@#A
6	7	relative	A
7	7	relative deferred	@A

1.5.2.10. Immediate mode

Het # -teken wordt in de assemblertalen gebruikt om aan te geven, dat hetgeen achter dit teken volgt, als getal geïnterpreteerd moet worden. Na de vertaling bestaat de instructie uit twee computer woorden. In het eerste woord wordt aangegeven dat het adres berekend moet worden met behulp van de programmateller via een auto increment mode. De programmateller wijst ondertussen al naar het volgende woord, zodat na het voltooien van de "auto increment mode" weer naar een volgende instructie wordt gewezen.

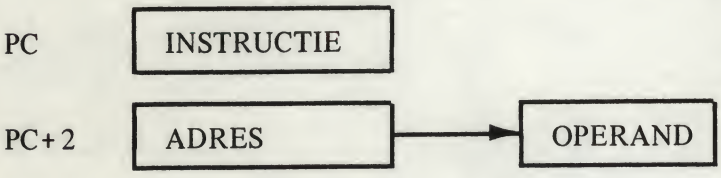


Voorbeeld: MOV # 2, Ro

Betekenis: Het getal 2 wordt in register 0 geplaatst.

1.5.2.11. Absolute mode

Voor deze mode geldt vrijwel hetzelfde als voor de immediate mode, alleen wordt het getal gebruikt om een operand aan te wijzen.



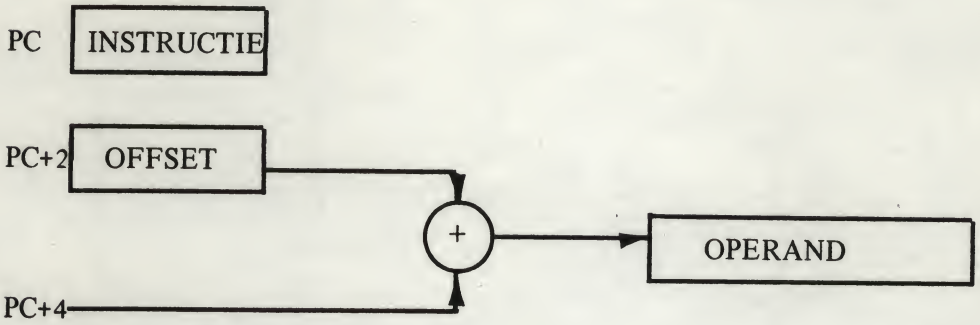
Voorbeeld: CLR @ #2000

Betekenis: De "absolute" locatie 2000 wordt nul gemaakt.

5.2.12. Relatieve mode

Met behulp van deze adresseermode wordt het mogelijk gemaakt gebruik te maken van symbolisch adresseren. De machinecode zal voor deze mode weer uit twee computerwoorden bestaan. Instructies die gebruik maken van de

relatieve mode, worden door de assembler vertaald als zijnde een indexed adresseermode via de programmateller. Bij de vertaling wordt in het tweede woord van de instructie een getal geplaatst, dat aangeeft hoe groot de offset is tot het symbolische adres.

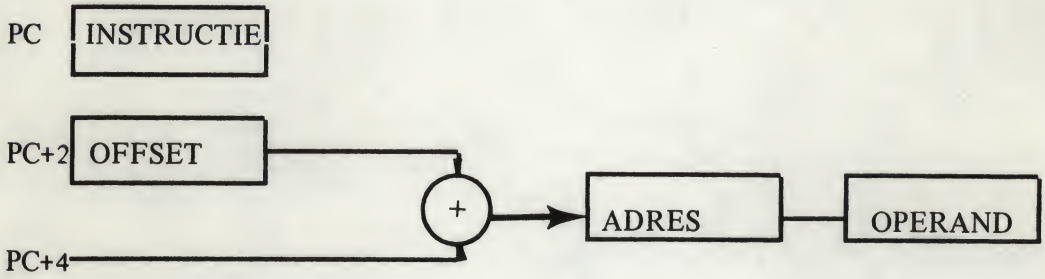


Voorbeeld: MOV A,B

Betekenis: De inhoud van de locatie met de symbolische naam A wordt ook geplaatst in de locatie met de symbolische naam B.

1.5.2.13 Relative Deferred Mode

Voor deze adresseermode geldt hetzelfde als voor de relatieve mode, met dit verschil, dat de inhoud van het symbolische adres gebruikt wordt als adres van de operand en niet als operand.

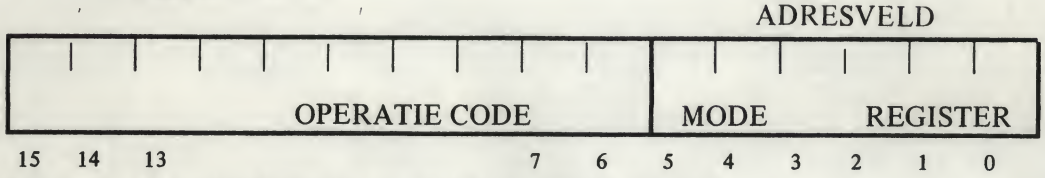


Voorbeeld: CLR @ A

Betekenis: De locatie waarvan het adres gevonden wordt in het symbolische adres A wordt nul gemaakt.

1.5.3. Enkel operand instructies

De enkel operand instructies is opgebouwd uit twee gedeelten, een 10 bits instructie code veld en een 6 bits adres veld.
De indeling is in figuur 7 in beeld gebracht. In het adresveld is met behulp van de adresseermodes het destination adres geplaatst. Het bestaat uit een 3 bits mode veld en een 3 bits registerveld. Een destination adres is het adres van de locatie waarop de bewerking plaats heeft.



figuur 7 – Woordindeling bij een enkel operand instructie

Bij instructies die zowel op een byte als op een woord betrekking kunnen hebben, geeft bit 15 aan of het een woord (bit 15 = 0) of een byte (bit 15 = 1) instructie betreft. Er volgt nu een lijst met enkel operand instructies. Hierin betekent CLR(B) dat de clear opdracht zowel op een woord als op een byte betrekking kan hebben. De code voor een woord instructie is CLR, voor een byte instructie CLRB.

Bij de machine code betekent:

x 1° bit kan 0 zijn bij een woord instructie
en 1 zijn bij een byte instructie
DD destination adresveld
SS source adresveld

Bij de conditie bits betekent:

x wordt gezet afhankelijk van het resultaat
- geen effekt.

Enkel operand instructies:

MNEMONIC OPCODE	MACHINE CODE	BETEKENIS DESTINATION WORDT:	CONDITIE BITS: N	Z	V	C	Opmerking:
CLR(B)	x050DD	nul	0	1	0	0	
COM(B)	x051DD	1 complement	x	x	0	1	
INC(B)	x052DD	destination + 1	x	x	x	-	
DEC(B)	x 53DD	destination - 1	x	x	x	-	
NEG(B)	x054DD	negatief destination	x	x	x	x	
TST(B)	x057DD	test destination	x	x	0	0	
ROR(B)	x060DD	schuif rond rechts	x	x	x	x	
ROL(B)	x061DD	schuif links rond	x	x	x	x	
ASR(B)	x062DD	schuif rekenkundig rechts	x	x	x	x	
ASL(B)	x063DD	schuif rekenkundig links	x	x	x	x	
SWAB	0003DD	verwissel bytes	x	x	x	0	
ADC(B)	x055DD	destination + carry	x	x	x	x	
SBC(B)	x056DD	destination - carry	x	x	x	x	
SXT	0067DD	teken uitbreiding	-	x	x	-	alleen LSI 11

Toelichting bij de schuifopdrachten:

Er bestaan, zoals in de instructielijst te zien is, twee soorten schuifopdrachten, ten eerste logische (rond) schuifopdrachten en ten tweede rekenkundige (arithmetische) schuifopdrachten. De verschillen in beide soorten schuifopdrachten worden hieronder aangegeven.

1o. Logische rondschuifopdrachten ROR(B) en ROL(B).

Tengevolge van een ROR instructie wordt de inhoud van het destinationadres één plaats naar rechts geschoven. Het meest linkse bit wordt hierbij aangevuld met de inhoud van het carry bit, terwijl de inhoud van het meest rechts bit weer in het carry bit geplaatst wordt. Er wordt dus rondgeschoven via het carry bit. Bij de ROL instructie vindt het omgekeerde plaats, zodat de inhoud van het destination adres een plaats naar links wordt geschoven. Het meest rechtse bit wordt aangevuld met de inhoud van het carry bit, terwijl de inhoud van het meest linkse bit in het carry bit geplaatst wordt.

LSI11 PROCESSOR STATUS WOORD INSTRUCTIES: MFPS 1067DD MOVE FROM PS

MTPS 1064SS MOVE TO PS

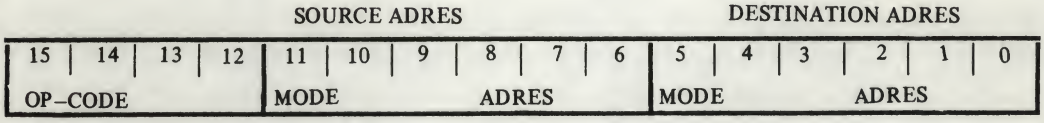
2o. Rekenkundige schuifopdrachten ASR(B) en ASL(B)

Bij de rekenkundige schuifopdracht ASR wordt de inhoud van het destination adres één plaats naar rechts geschoven. De inhoud van het meest rechtse bit wordt in het carry bit geplaatst. Het meest linkse bit blijft echter ongewijzigd. Dit betekent dat het tekenbit van het in behandeling zijnde woord intact blijft. Het rekenkundig schuiven naar rechts komt overeen met delen door twee.

Bij de rekenkundige schuifopdracht ASL wordt de inhoud van het destination adres één plaats naar links geschoven. Het meest rechtse bit wordt nu gevuld met nul. De inhoud van het meest linkse bit wordt in het carry bit geplaatst. Het rekenkundig naar links schuiven komt overeen met vermenigvuldigen met twee, waarbij ook het overflow bit V op de juiste manier gezet wordt.

1.5.4. Dubbel operand instructies

De dubbel operand instructies bevatten naast de operatiecode twee adresvelden, één voor de source operand (waar de data vandaan komt) en één voor de destination operand (waar de data naar toe moet). In figuur 8 is de woordopbouw van een dubbel operand instructie in beeld gebracht. De operatie code bestaat uit 4 bits en de adresvelden elk uit 6 bits.



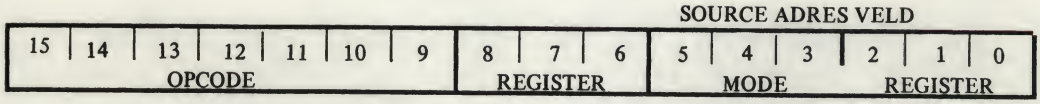
figuur 8 – Woordindeling bij een dubbel operand instructie

Bij instructies, die zowel op woorden als op bytes betrekking kunnen hebben, geeft bit 15 aan of het een woord of een byte instructie betreft. Er volgt een lijst met dubbel operand instructies. Voor de verklaring van de gebruikte tekens wordt verwezen naar de vorige paragraaf.

Dubbel operand instructies

MNEMONIC OPCODE	MACHINE CODE	BETEKENIS IN DE VORM DESTINATION WORDT:	CONDITIE BITS N	Z	V	C	Opmerking:
MOV(B)	x1 SSDD	source	x	x	o	-	
CMP(B)	x2 SSDD	Set cond. bits volgens source-destination	x	x	x	x	
ADD	06 SSDD	source + destination	x	x	x	x	
SUB	16 SSDD	destination - source	x	x	x	x	
BIT(B)	x3 SSDD	Set conditie bits volgen source Λ destination	x	x	o	-	AND
BIC(B)	x4 SSDD	Clear bits in destination volgens source	x	x	o	-	
BIS(B)	x5 SSDD	Set bits in destination volgens source	x	x	o	-	OR

Voor de LSI 11 met hardware rekenfaciliteiten bestaan nog de volgende dubbel operand instructies waarbij het resultaat niet in een destination adres, doch in een register wordt opgeborgen. De woordindeling van deze instructies wijkt dan ook af van de eerder genoemde en is weergegeven in figuur 9.



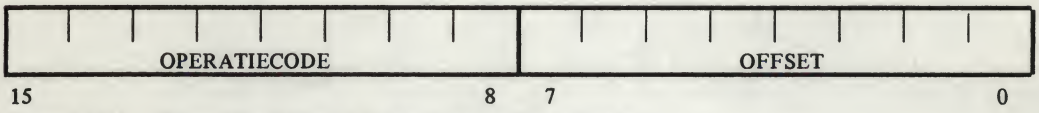
figuur 9 – Woordindeling bij een instructie voor de hardware rekenunit EIS (alleen LSI11)

MNEMONIC OP CODE	MACHINE CODE	BETEKENIS	CONDITIE BITS N Z V C	Opmerking
MUL	070RSS	Register:= Register x source	x x o x	alleen LSI
DIV	071RSS	Register:= Register / source	x x x x	alleen LSI
ASH	072RSS	meervoudig schuiven	x x x x	alleen LSI
ASHC	073RSS	gecombineerd schuiven	x x x x	alleen LSI
XOR	074RDD	exclusieve OR	x x o -	alleen LSI

Voor een nadere beschrijving van de hardware rekeninstructieset wordt verwezen naar het processor handbook van DEC.

1.5.5. Sprong instructies

Met spronginstructies is het mogelijk de executie volgorde van de instructies te veranderen. De sprong instructies zijn onder te verdelen in: conditionele en onvoorwaardelijke sprongen. Conditionele sprongen worden uitgevoerd afhankelijk van de inhoud van een of meer conditiecodebits uit het processor status woord. De spronginstructies zijn opgebouwd uit twee gedeelten, een instructiecodeveld en een sprongoffset veld. Een uitzondering hierop wordt gevormd door de JMP instructie, die dezelfde opbouw heeft als een enkel operand instructie en waarbij gesprongen wordt naar het destination adres. De opbouw van de spronginstructies is weergegeven in figuur 10.



figuur 10 – Woordindeling bij een spronginstructie

Het offsetgebied bestaat uit 8 bits, waaruit volgt dat sprongen, m.u.v. de JMP opdracht, niet plaats kunnen vinden over grotere afstanden dan $\pm 2^7$ woorden. Dit betekent dat niet meer dan 127 plaatsen voorwaarts en niet meer dan 128 plaatsen achterwaarts gesprongen kan worden.

Sprong instructies: *Conditie codes blijven ongewijzigd*

MNEMONIC OP-CODE	MACHINE CODE	BETEKENIS	CONDITIES
JMP	0001DD	Spring altijd	
BR	000400+xxx	Spring altijd	
BNE	001000+xxx	Spring bij niet gelijk	Z = 0
BEQ	001400+xxx	Spring bij gelijk	Z = 1
BPL	100000+xxx	Spring indien positief	N = 0
BMI	100400+xxx	Spring indien negatief	N = 1
BVC	102000+xxx	Spring indien V = 0	V = 0
BVS	102400+xxx	Spring indien V = 1	V = 1
BCC	103000+xxx	Spring indien C = 0	C = 0
BCS	103400+xxx	Spring indien C = 1	C = 1
BGE	002000+xxx	Spring bij groter of gelijk 0	N ⊻ V = 0
BLT	002400+xxx	Spring indien kleiner dan 0	N ⊻ V = 1
BGT	003000+xxx	Spring indien groter dan 0	Z v (N ⊻ V) = 0
BLE	003400+xxx	Spring indien kleiner of gelijk 0	Z v (N ⊻ V) = 1
BHI	101000+xxx	Spring indien groter	C v Z = 0
BLOS	101400+xxx	Spring indien kleiner of gelijk	C v Z = 1
BHIS	103000+xxx	Spring indien groter of gelijk	C = 0
BLO	103400+xxx	Spring indien kleiner	C = 1

Een speciale, alleen op de LSI11 aanwezige instructie is de SOB instructie. De betekenis is de volgende:

SOB	077RNN	Register R wordt met 1 verlaagd en een sprong naar PC−(2xNN) wordt uitgevoerd indien R ≠ 0 is.

1.5.6. Conditiecodeinstructies

Conditiecodeinstructies zijn instructies waarmee het mogelijk is de conditie bits uit het processor status woord te beïnvloeden. Hiervan kan bijvoorbeeld gebruik worden gemaakt bij het verlaten van een subroutine om aan te geven dat de verwerking binnen de subroutine al dan niet correct is geweest.

Conditiecodeinstructies

MNEMONIC OPCODE	MACHINE CODE	BETEKENIS	CONDITIE BITS N Z V C
CLC	000241	Clear C-bit	- - - o
CLV	000242	Clear V-bit	- - o -
CLZ	000244	Clear Z-bit	- o - -
CLN	000250	Clear N-bit	0 - - -
SEC	000261	Set C-bit	- - - 1
SEV	000262	Set V-bit	- - 1 -
SEZ	000264	Set Z-bit	- 1 - -
SEN	000270	Set N-bit	1 - - -
SCC	000277	Set all bits	1 1 1 1

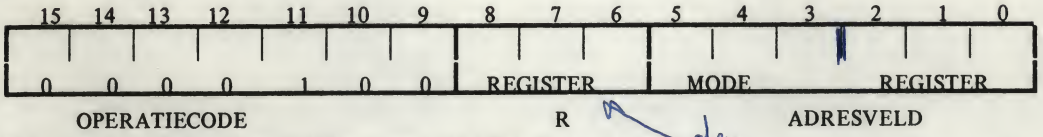
1.5.7. Subroutine instructies

Voor het gebruik van subroutines zijn twee instructies aanwezig.

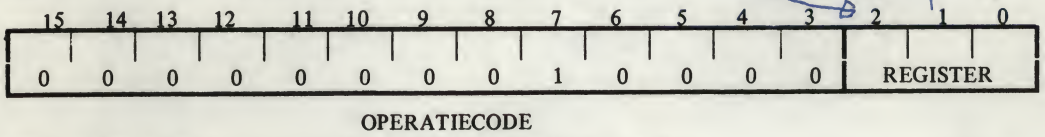
1°. De subroutine sprong – JSR – 004RDD

2°. De terugkeer uit een subroutine – RTS – 00020R

In figuur 11 is in indeling van een JSR instructie weergegeven, in figuur 12 hetzelfde voor de RTS instructie.



Figuur 11 – De woordindeling van de JSR instructie



Figuur 12 – De woordindeling van de RTS instructie

Bij de subroutine aanroep wordt een register gebruikt om het terugkeeradres, d.w.z. het adres waar het aanroepende programma na afhandeling van de subroutine moet worden voortgezet, te onthouden. Dit mag register 7 de PC zijn, hetgeen dan de efficientste oplossing levert, omdat geen extra register benodigd is. Wordt echter binnen de subroutine van de terugkeer locatie gebruik gemaakt, dan moet gebruik gemaakt worden van een der registers 0 t/m 5.

Bij de terugkeer uit de subroutine moet van hetzelfde register gebruik gemaakt worden als bij de aanroep. Bij een subroutine aanroep vinden binnen de machine achtereenvolgens de volgende handelingen plaats.

1°. De locatie, bepaald door het adresveld in de instructie, waarnaartoe de subroutine sprong plaats heeft wordt in een intern register TEMP opgeborgen.

TEMP ← (DST)

2°. De inhoud, van het bij de subroutinesprong gebruikte register wordt gered op de stack.

↓ (SP) ← reg

3°. Het terugkeer adres, dit is het adres waar op dit moment de programmateller naar wijst, wordt in het register geplaatst.

reg ← PC

4°. De sprong wordt uitgevoerd doordat de programmateller gevuld wordt met de inhoud van het interne register TEMP

PC ← TEMP

Bij de terugkeer uit de subroutine vindt binnen de machine achtereenvolgens plaats:

1°. De programma teller wordt gevuld met de inhoud van het bij de aanroep en terugkeer betrokken register, waarmee de terugkeer sprong uitgevoerd is

PC ← reg

2°. De oude inhoud van het bij de subroutine terugkeer instructie betrokken register wordt van de stack gehaald en in het register teruggeplaatst.

reg ← (SP)↑

1.5.8. Operatie instructies

Operatie instructies zijn instructies waarmee commando's aan de machine kunnen worden gegeven. De instructies bestaan enkel uit een operatiecode.

Operatie instructies:

MNEMONIC OPCODE	MACHINE CODE	BETEKENIS
HALT	000000	Stop executie
WAIT	000001	Wacht op een interrupt
RESET	000005	Reset het systeem
NOP	000240	geen operatie

1.5.9. Floating point instructies (Alleen voor LSI 11)

Voor de LSI 11 computer, die uitgerust is met hardware floating point rekenfaciliteiten, bestaan een aantal floating point instructies. De woordindeling van de instructie is gelijk aan die van de RTS (return van subroutine) instructie. De bij de bewerking behorende floating point getallen dienen aaneengesloten in een gebied geplaatst te worden. Het bij de instructie gebruikte register moet het adres van de eerste locatie van dit gebied bevatten. Het resultaat van de bewerking wordt in hetzelfde gebied geplaatst.
Voorbeeld: De getallen A en B zijn bij een floating point bewerking betrokken. Het gebied waarin A en B geplaatst zijn moet er nu als volgt uitzien:

R →	B_2	High B argument address	(R)
	B_1	Low B argument address	(R)+2
	A_2	High A argument address	(R)+4
	A_1	Low A argument address	(R)+6

Het register R wijst naar de eerste locatie van het gebied.

Na de bewerking is het resultaat geplaatst op de twee laatste locaties, terwijl R nu naar het eerste van deze twee locaties wijst. (vergelijk LIFO stack).

R ⇢	B_2		
	B_1		
R →	C_2	High result address	(R)
	C_1	Low result address	(R)+2

Floating point bewerkingen (Alleen LSI 11)

MNEMONIC OP CODE	MACHINE CODE	BETEKENIS	CONDITIE BITS N Z V C
FADD	07500R	floating optellen	x x o o
FSUB	07501R	floating aftrekken	x x o o
FMUL	07502R	floating vermenigvuldigen	x x o o
FDIV	07503R	floating delen	x x o o

22-11-78

1.5.10 Interrupties

Alvorens in te gaan op de interruptie instructies wordt aandacht besteed aan de interruptie structuur van de PDP11. We onderscheiden drie soorten interrupties, die alle op dezelfde manier afgehandeld worden:

1°. Geprogrammeerde interrupties (z.g. TRAPS)
2°. Interrupties door randapparatuur
3°. Interrupties door hardware fouten.

Op al deze soorten interrupties wordt verder op teruggekomen.

1.5.10.1. Interruptie structuur

Op een bepaald moment kan slechts één device als unibus master optreden. Dit master device wisselt dan informatie uit met een slave device, dat ook verbonden is met de unibus. Deze informatie kan bestaan uit instructies, adressen of data. Een typisch voorbeeld van deze informatie overdracht treedt op, als de central processor als busmaster instructies, data of operanden weghaalt uit het geheugen (slave).

Wanneer een randapparaat door middel van een bus request vraagt de unibus te mogen gebruiken, dan gebeurt dit in het algemeen om een van de volgende twee redenen:

1. Het doen van een *"Non Processor Transfer"*. Hieronder wordt verstaan, het overbrengen van datagegevens van het ene naar het andere apparaat zonder tussenkomst van de central processor.

2. Het plegen van een *"interrupt"* waardoor de executie van het lopende programma tijdelijk wordt onderbroken voor de afhandeling van de bij de interruptie behorende service routine.

Een bus request van een randapparaat kan plaatsvinden via één van de vijf request lijnen: De non processor request lijn of de busrequest lijnen 7 t/m 4. Hierin heeft de non processor request lijn de hoogste prioriteit. De prioriteit van de overige lijnen neemt af met het nummer van de lijn.

Aan elk randapparaat wordt in hardware een interruptie niveau toegekend terwijl de central processor een programmeerbaar prioriteitsniveau heeft (Zie processor status woord).

Een interruptie wordt alleen in behandeling genomen wanneer de bijbehorende prioriteit groter is dan de prioriteit waarop de central processor op dat moment werkt.

1.5.10.2. Non Processor Request

Zoals hiervoor reeds opgemerkt kan een data transport tussen randapparaten plaatsvinden zonder tussenkomst van de central processor. Op het moment dat een non processor request wordt gedaan , wordt de lopende unibus cyclus afgemaakt, waarna de non processor request behandeld wordt. De non processor request wordt toegepast voor snel data transport tussen bijvoorbeeld een schijfgeheugen en het werk-geheugen. Er wordt dan gebruik gemaakt van een DMA (Direct Memory Access) interface.

1.5.10.3. Interrupt Request

Als een apparaat een bus request op een van de lijnen 7 t/m 4 plaatst dan zal dit apparaat busmaster worden onder voorwaarde dat het nummer van de bus request lijn hoger is dan het interrupt niveau van het device dat op dat moment bus master is. (Eventueel de central processor). Indien aan deze voorwaarde voldaan is, wordt de nieuwe bus master in de gelegenheid gesteld het adres van de *interrupt vektor* naar de centrale processor te zenden. Voor elk apparaat zijn namelijk in het core memory twee geheugenplaatsen – de interruptvektor – gereserveerd. In deze geheugenplaatsen dient door de programmeur vermeld te worden:

- Het startadres van de interrupt service routine
- De inhoud van het processor status register gedurende de executie van de service routine (Prioriteit).

Het adres van de interrupt vektor is vastgelegd in de interface van het randapparaat. Op het moment, dat de interrupt veroorzaakt wordt zal de central processor eerst de lopende instructie nog helemaal uitvoeren. Dit in tegenstelling tot hetgeen gebeurt bij een non-processor request. Bij een interrupt gebeurt er achtereenvolgens het volgende:

- de lopende instructie wordt afgemaakt
- het vektor adres van het randapparaat wordt naar de central processor gezonden
- de inhoud van het processor status register en de programmateller worden op de stack gered om na afhandeling van de interruptie het programma te kunnen vervolgen
- de nieuwe programma teller en processor status woord inhoud wordt uit de interrupt vektor gehaald
- na afloop van de interruptie dient met behulp van een RTI instructie naar het onderbroken programma teruggekeerd te worden. Hierdoor worden op de stack geplaatste programma teller en proces status register inhoud teruggeplaatst. Daarna is het onderbroken programma vervolgd.

OPMERKING:

Een interruptie door een randapparaat ontstaat wanneer dat randapparaat een gereedmelding geeft èn wanneer van dat apparaat door middel van het 1 maken van het interrupt enable bit in het status register een interruptie gewenst wordt.

1.5.10.4. Geprogrammeerde interrupts - TRAPS

Traps kunnen opgevat worden als in een programma gegenereerde interrupts. Het interruptie niveau van de traps is hoger dan de hoogste prioriteit, die aan een randapparaat kan worden gegeven.

Evenals bij de normale interruptie, wordt ook bij een trap de programma teller en het processor status register geladen vanuit een trapvektor, nadat de oorspronkelijke inhoud op de stack geplaatst is.

Ook traps hebben tot gevolg, dat er een trapservice routine afgewerkt wordt, welke tevoren door de programmeur in het geheugen is geplaatst.

1.5.10.5. Hardware fouten – TRAPS

Naast de geprogrammeerde intertupties bestaan nog interrupties ten gevolge van hardware fouten, zoals het programmeren in een niet bestaand geheugen gedeelte en het loslaten van een woordinstructie op een oneven byte locatie. Ook dit soort interrupties worden traps genoemd. De bijbehorende vektor adressen liggen in de machine vast en zijn resp.:

VEKTORADRES:	INTERRUPTIE OORZAAK:
4	Timeout (en andere) - adressering niet bestaande locatie
10	illegale instructie.

1.5.10.6. Interruptie instructies

– *Return from interrupt – RTI*
Mnemonic Opcode: RTI
Machine code : 000002

Deze instructie **moet** gebruikt worden om na afhandeling van een interruptie terug te keren naar het onderbroken programma. Als gevolg van deze instruktie worden de programmateller en het processor status register weer gevuld vanuit de stack.

– *Emulator trap instructie – EMT*
Mnemonic Opcode: EMT xxx
Machine code: 104000+xxx
Interruptie vektor: 30

Deze instruktie zal een trap veroorzaken met als trap vektor adres locatie 30. In het xxx gedeelte van de instructie kan een code meegegeven worden. Dit betekent dat alle operatiecodes van 104000 tot 104377 representaties van EMT instructies zijn.

De EMT instructie wordt binnen het RT11 operating systeem gebruikt door de monitor, en is dus niet voor vrij gebruik beschikbaar.

– *Trap instructie – TRAP*
Mnemonic opcode: TRAP xxx
Machine code: 104400+xxx
Interruptie vektor: 34

Voor deze instruktie geldt hetzelfde als voor de EMT instructie, met dit verschil dat de trapvektor in geheugen plaats 34 is ondergebracht. De machine codes 104400 t/m 104777 zijn alle geldige representaties van TRAP instructies.

– *Input Output trap instructie – IOT*
Mnemonic opcode: IOT
Operatiecode: 000004
Interruptie vektor: 20

Voor deze instruktie geldt hetzelfde als voor de EMT instruktie, met dit verschil, dat in deze instructie geen informatie voor de service routine geplaatst kan worden.

– *Trap voor ODT – BPT*
Mnemonic opcode: BPT
Operatie code: 000003
Interruptie vektor: 14

Voor deze instruktie geldt hetzelfde als voor de IOT-instruktie. In het ODT-programma wordt van deze Break Point Trap instruktie gebruik gemaakt.

1.6. Randapparatuur

Voor het programmeren van de randapparatuur van de PDP-11 is geen speciale klasse van in- en uitvoerinstrukties aanwezig. De Unibus waarop zowel de Central Processor als alle randapparatuur zijn aangesloten, biedt de mogelijkheid tot een uniforme adresserings-struktuur, waarin Control-Status- en Data registers van alle aangesloten randapparatuur direkt als geheugenplaatsen geadresseerd kunnen worden.

Doordat het mogelijk is gebruik te maken van alle instrukties op de randapparatuur registers ontstaat een zeer flexibele input/output programmering. Alle randapparatuur registers kunnen behandeld worden als **accumulatoren,** zodat het niet nodig is de hierin aanwezige informatie te verplaatsen alvorens rekenkundige operaties of vergelijkingen uit te voeren.
Elk randapparaat bezit meestal twee registers:

1. Control-Status Register
2. Data Register

In figuur 1 is een beeld gegeven van de inhoud van een Control-Status Register. De register adressen van de randapparaten bevinden zich in het gebied 160000 t/m 177776, octaal genummerd, hetgeen overeen zou komen met het geheugengebied 28 t/m 32 k woorden.

1.6.1. Teletype/terminal

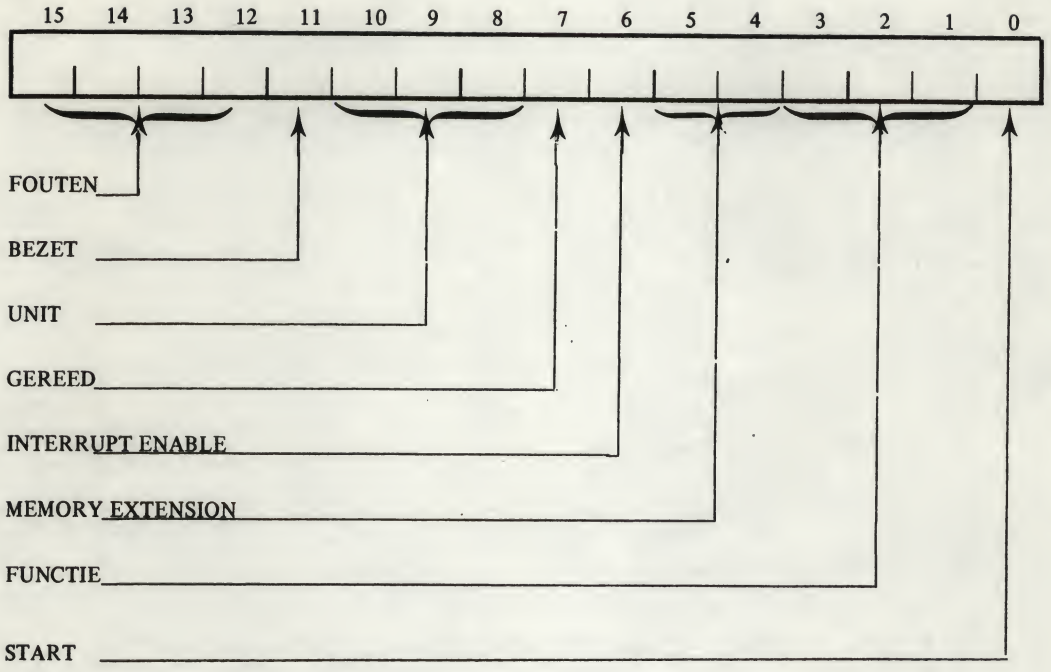
De teletype bestaat uit:

Keyboard/Ponsbandlezer
Printer/Ponsbandponser.

Door de computer worden Keyboard en lezer evenals Printer en Ponser als één apparaat gezien, waarbij het van de schakelaars op de Teletype afhangt welk apparaat in functie is.
Een terminal bestaat alleen uit een keyboard en een printer.

Printer/Ponsbandponser

Deze uitvoerapparaten worden als één apparaat gezien. Door middel van de keuzeschakelaar op de Teletype kan de ponser aan of uitgeschakeld worden.



figuur 1. – Voorbeeld Control-Status Register

Register adressen: Status Register : 177564 (TPS)
Data Register : 177566 (TPB)
Interrupt vektor adres: 64 en 66.

Het device wordt aangeroepen door informatie in het dataregister (TPB) te plaatsen, bijvoorbeeld met behulp van een MOV instruktie. Deze informatie wordt dan direkt getypt (eventueel ook geponsd). Tijdens het uitvoeren is het Ready-bit (bit 7) van het Status Register (TPS) nul. Wanneer het device klaar is met het uitvoeren van de aangeboden informatie wordt het Ready bit geset.
Alvorens voor een volgende uitvoer actie informatie aangeboden wordt, moet getest worden of de vorige uitvoeractie beëindigd is. Hiertoe moet getest worden of het Ready-bit van het Status Register een is. (Dit kan door het teken te bepalen van het low-order byte van het Status Register).
Wanneer asynchroon met de printer/ponser gewerkt wordt, zal zodra het Readybit van het Status Register één wordt, een interrupt ontstaan. Hiertoe moet in het Status Register het interrupt Enable bit (bit 6) geset zijn en moeten in de locaties 64 en 66 resp. het adres van de Interrupt routine en het nieuwe prioriteitsniveau gegeven zijn.

Keyboard / Ponsbandlezer

Keyboard en ponsbandlezer worden door de computer als één invoer apparaat gezien, waarbij het aan of uitstaan van de ponsbandlezer bepaalt welk invoer apparaat in bedrijf is.

Register adressen: Status Register : 177560 (TKS)
Data Register : 177562 (TKB)
Interrupt vektor adres : 60 en 62

Wanneer de lezer gebruikt wordt moet een start commando voor elk te lezen karakter gegeven worden. Dit gebeurt door het start bit (bit 0) van het Status Register één te maken.
Is de leesactie beëindigd, of ingeval er keyboard karakters ingevoerd worden, is er een karakter getypt, dan wordt het Ready bit (bit 7) in het Status Register hoog. Dit betekent dat de informatie gereed staat in het Data-Regiser.
Met behulp van b.v. een MOV instruktie kan de informatie nu overgenomen worden. De informatie mag niet eerder overgenomen worden, dan het Ready bit geset is. Hier kan op getest worden door het low-order byte van het Status Register te testen. Zolang het teken van dit byte positief is, is de invoeractie nog niet beëindigd. Als de informatie overgenomen is kan een volgende leesactie gestart worden.

Wordt asynchroon met het keyboard/ponsbandlezer gewerkt, dan ontstaat als het Ready bit één wordt een interrupt. Hiertoe moet in het Status Register het Interrupt Enable bit (bit 6) geset zijn en moeten in de locaties 60 en 62 resp. het adres van de interrupt routine en het nieuwe prioriteitsniveau gegeven zijn.

Programma voorbeelden:

Voorbeeld 1: *Uitvoeren van een karakter naar Printer/Ponser*

```
      TPS=177564        Declaratie Printer Status Register
      TPB=177566        Declaratie Printer Data Register
TEST: TSTB TRS          TEST of READY BIT 1 is
      BPL TEST          zo nee, spring naar TEST
      MOVB KAR, TPB     UITVOER OPDRACHT
```

Voorbeeld 2: *Initiërende acties voor Interrupt*

```
      .=64              Vullen interrupt verkort met
      PRINT             begin adres interrupt routine en
      200               bijbehorend prioriteitsniveau
       .
       .                programma
       .
      MOV # 100, TPS    Interrupt Enable
       .
```

Voorbeeld 3: *Het lezen van karakters*

```
      TRS=177560        Declaratie Reader Status Register
      TRB=177562        Declaratie Reader Data Register
      INC TRS           Start leesactie
TEST: TSTB TPS          Test of Ready Bit
      BPL TEST          Spring naar TEST bij niet ready
      MOVB TRB,DATA     Overnemen ingelezen karakter
```

1.6.2. Snelle ponsband lezer/ponser

De programmering van de snelle ponsband lezer/ponser is volkomen identiek aan die van de ponsband lezer/ponser van de teletype. Alleen de Status en Data Registers hebben andere adressen. Voor de programmering wordt dan ook naar 1.6.1. verwezen.

Register adressen:

Ponsband lezer:	Status Register:	177550	(PRS)
	Data Register :	17752	(PRB)
	Interrupt Vektor adres: 70 en 72.		
Ponsbandponser:	Status Register :	177554	(PPS)
	Data Register :	177556	(PPB)
	Interrupt vektor adres: 74 en 76.		

1.6.3. AD-converter (alleen voor PDP 11/20)

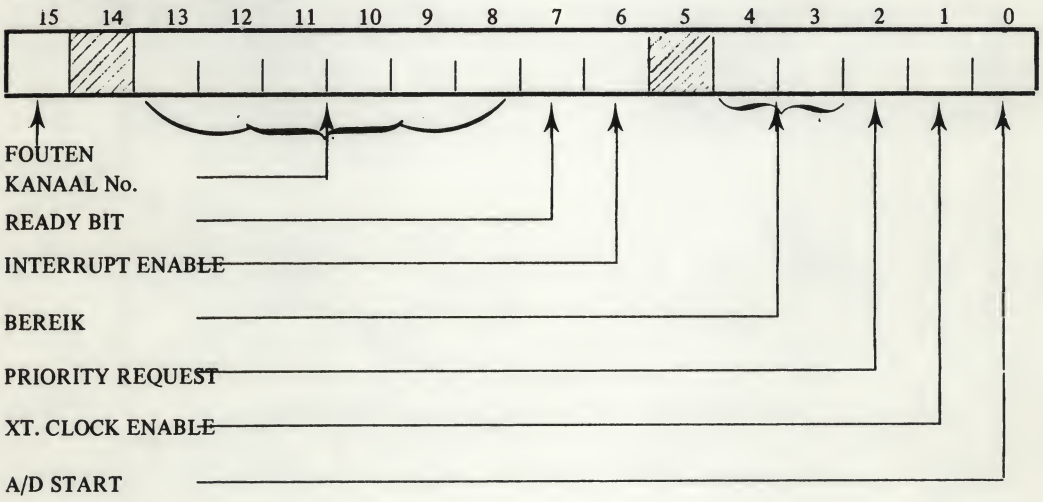
AD 01

De aanwezige AD converter heeft een resolutie van 11 bits, vier programmeerbare bereiken en een conversietijd van 30 μsec. Voor de programmering zijn aanwezig een Control en Status Register en een Data Register.

Register adressen:

Control en Status register:		176770
Data Register	:	176772
Interrupt vektor adres	:	130 en 132.

Control/Status Register



Verklaring Control en Status Register

Bit	Verklaring
15	Wordt 1 als een nieuwe conversie gestart terwijl de vorige nog niet afgewerkt is
13–8	Kanaalnummers voor Multiplexer
7	Ready bit – gereedmelding
6	Door bit 6 hoog te maken ontstaat interrupt mode, bij gereedmelding ontstaat interrupt naar het adres aangegeven in locatie 130
4–3	Bepalen het ingangsbereik

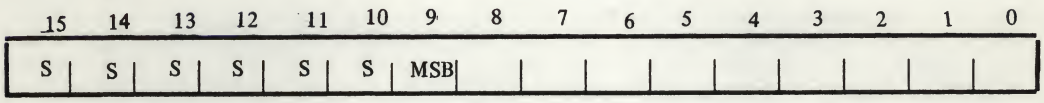
Overzicht bereiken:

bit 4	bit 3	bereik
0	0	± 10V
0	1	± 5V
1	0	± 2 ½V
1	1	± 1¼ V

1 Externe clock enable: De AD converter kan door een externe clock gestuurd worden. Hiertoe moet een extern clocksignaal aan de AD converter aangesloten worden en moet bit 1 hoog gemaakt worden.

0 Start A/D conversie als bit 0 geset wordt (De conversie wordt ook gestart als een nieuw multiplexer kanaal of een nieuw bereik wordt aangeboden)

Data Register



Bit 15–10 Tekenuitbreiding
Bit 9– 0 Informatie, bit 9, most significant bit.

Enkele voorbeelden van omgezette spanningen uitgaande van het bereik ±10V:

– 10.0 V	176000
– 5.0 V	177000
0 V	000000
5 V	001000
+9.9902 V	001777

Programma voorbeeld:

```
        ADCSR = 176770          ; Declaratie Status Register
        ADDB  = 176772          ; Declaratie Data Register
        MOV # 400,ADCSR         ; Selecteer kanaal 1
                                        bereik ±10V, Start AD conversie
TEST:   TSTB  ADCSR
        BPL   TEST              ; Test op gereed
        MOV ADDB, DATA          ; Neem data over
```

1.6.4. DA-converter/Display Scope (Alleen PDP11/20)

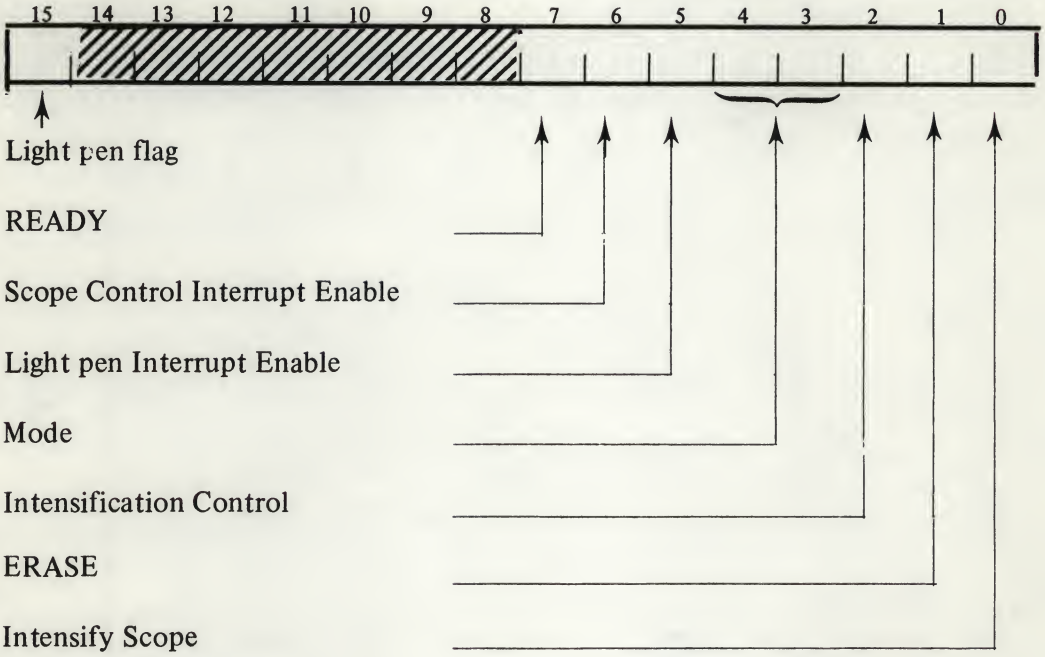
Algemene gegevens:

Woordlengte : 12 bits bipolair
Bereik : instelbaar; max. ± 10V/min. ± IV
Belasting : max. 10 mA

Register adressen:

Control/Status Register	(CSR)	176756
Data Register 0	(DAC 0)	176760
Data Register 1	(DAC 1)	176762
Data Register 2	(DAC 2)	176764
Data Register 3	(DAC 3)	176766

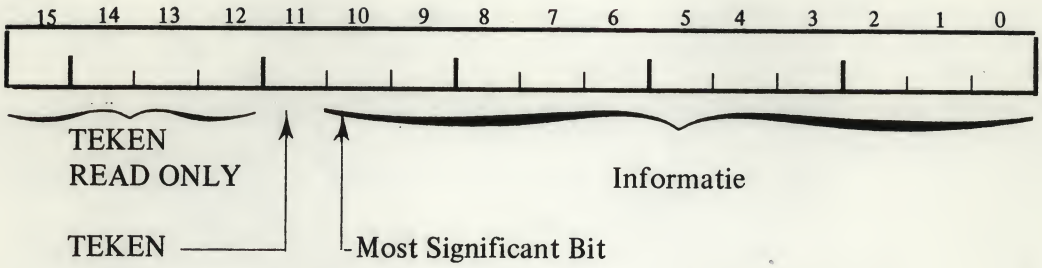
Overzicht Control/Status Register:



Verklaring Control en Status Register

Bit	Verklaring
15	t.b.v. lichtpen (niet aanwezig)
14–8	Ongebruikt
7	Gereed bit
6	Interrupt Enable (alleen voor Scope Control)
5	Interrupt Enable (alleen voor lichtpen)
4–3	Mode Control (Intensification bij het vullen van de X of de Y coördinaat).
2	Storage/Non storage (voor scope control)
1	Erase (voor scope control)
0	Intensification (voor scope control)

Data Register



Algemeen

Om informatie naar een bepaald kanaalnummer uit te voeren, moet de informatie in de betreffende DAC buffer geplaatst worden. Hierna vindt, zonder verdere commando's direkt de conversie plaats.
Voor scope display zijn enkele extra commando's aanwezig.
Wil men een punt op de scoop oplichten, dan moeten eerst de x-coördinaat en de y-coördinaat aan de DAC kanalen 0 en 1 aangeboden worden.
Hierna kan door een intensity commando (bit 0 hoog maken) het punt oplichten.
Afhankelijk van bit 2 blijft het punt oplichten (in storage mode – bit 2 hoog) of dooft direkt weer (non storage mode.).
Ook kan men het punt laten oplichten na aanbieding van de X of de Y formatie. (bit 3.4)

Programmeer voorbeelden:

Voorbeeld 1:

Genereren van een zaagtand naar kanaal 2 DA converter.

```
CSR   = 176756                ;   Declaraties
DAC2  = 176764
        CLR     DAC 2
TEST:   TSTB    CSR           ;   TEST OP GEREED
        BPL     TEST          ;   NEE, SPRING NAAR TEST
        ADD     #100, DAC2    ;   VERGROOT SIGNAAL VAN DAC2
        BR      TEST          ;   SPRING NAAR TEST
```

Voorbeeld 2:

Uitvoeren van informatie naar de scope.

```
CSR   = 176756
DAC0  = 176760                ;   Declaraties                 Xas
DAC1  = 176762                ;                               Yas
        MOV INF, DACO         ;   Plaats X informatie in kanaal 0
        MOV INF, DACI         ;   Plaats Y informatie in kanaal 1
TEST:   TSTB    CSR               Test op gereed
        BPL     TEST
        MOV     #5, CSR       ;   Display commando, storage mode.
```

.
.
.

1.6.5. Digitale In- en Uitvoer.

De *digitale invoer* bestaat uit een 16 bits invoer apparaat met een gereed en een accept signaal.
Het gereed signaal (REQUESTB) wordt door de gebruiker aangeboden en geeft een teken aan de computer om de informatie op de 16 invoerlijnen over te nemen.
Wanneer de computer deze informatie overgenomen heeft, ontstaat een accept-signaal (puls) hetgeen voor de gebruiker een teken is om nieuwe informatie aan te bieden.
De *digitale uitvoer* bestaat uit een 16-bits uitvoer buffer met een gereed en een acceptsignaal.
Het gereedsignaal (puls) wordt door de computer gegeven wanneer de uitvoerbuffer gevuld is. De gebruiker kan nu een acceptsignaal (REQUESTA) geven ten teken dat de informatie verwerkt is en de volgende informatie uitgevoerd kan worden.

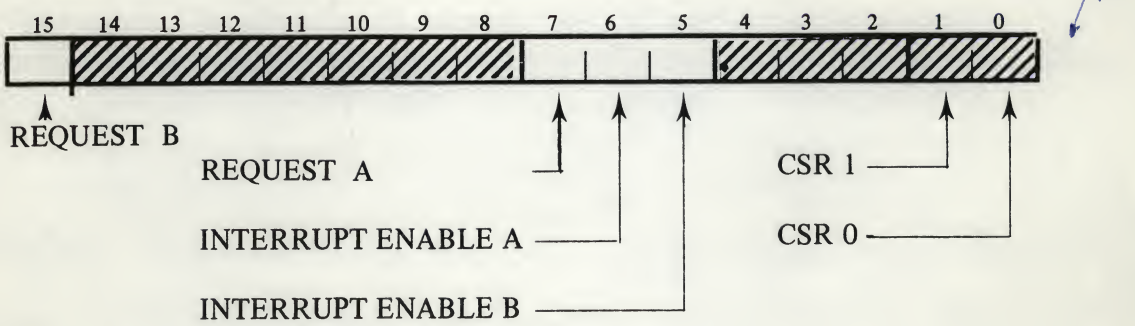
Programmering:

De totale Digitale In- en Uitvoer unit maakt gebruik van drie registers:

n.l.	167770	Statusregister
	167772	Uitvoer data register
	167774	Invoer data register

In het statuswoord is de toestand van de unit te vinden.
De vectoradressen zijn 300 en 304.

Overzicht Control/Status register



Overzicht signalen

REQ A : Dig. uitvoer – Extern aan te bieden signaal, ten teken dat de uit te voeren informatie verwerkt is en dat volgende informatie aangeboden kan worden. (Signaal moet door DREADY-puls gereset worden).

DREADY : Dig. Uitvoer – Puls uit de computer ten teken dat de computer nieuwe data in de uitvoer buffer geplaatst heeft.

REQ B : Dig. Invoer – Extern aan te bieden signaal ten teken dat in te voeren data gereed is (Signaal moet door DTRANS-puls gereset worden).

DTRANS : Dig. Invoer-puls uit de computer ten teken dat de in te voeren data overgenomen is en dat nieuwe informatie aangeboden kan worden.

Interrupties:

Het is mogelijk om de Digitale In- en Uivoer asynchroon te gebruiken. Hiervoor moet in het Status Register (adres 167770) bit 5 hoog gemaakt worden voor Invoer interruptie of bit 6 voor uitvoer interrupties.
Bij een interruptie vindt een sprong plaats naar een locatie aangegeven in het bijbehorende vectoradres.

Vectoradressen: Uitvoer vector A: 300
Invoer vector B : 304

Programma voorbeeld:

Invoer m.b.v. digitale invoer.

```
CSR   = 167770              ; declaratie statusregister
DIGIN = 167774              ; declaratie dataregister
TEST:   TSTB  CSR           ; Test op gereed, bit 7 hoog
        BPL   TEST
        MOV DIGIN, DATA     ; Overname informatie
```

1.6.6. Real Time Clock (alleen PDP11/20)

De aanwezige real time clock KW 11-P maakt het mogelijk op verschillende manieren real-time interrupts te genereren.
Hiertoe zijn aanwezig: een 16 bits teller, een 16 bits data buffer en een 7 bits control/status register.
Er kan gekozen worden uit vier verschillende telbronnen; 100 KHz generator, 10kHz generator, 50 hz netfrekwentie of een externe telbron.

De clock heeft twee verschillende operatiemodes:
Mode 0: Een ingesteld tijdsinterval wordt éénmaal doorlopen.
Mode 1: Een ingesteld tijdsinterval wordt telkens opnieuw doorlopen.
Een eventuele derde gebruiksmode is het tellen van externe gebeurtenissen.
Voor de programmering zijn de volgende registers aanwezig:

Control en Status Register	172540
Count Set Buffer Register	172542
Counter	172544
Interrupt Vektor Adres	104.
(Interruptie niveau BR6)	

Control and Status Register

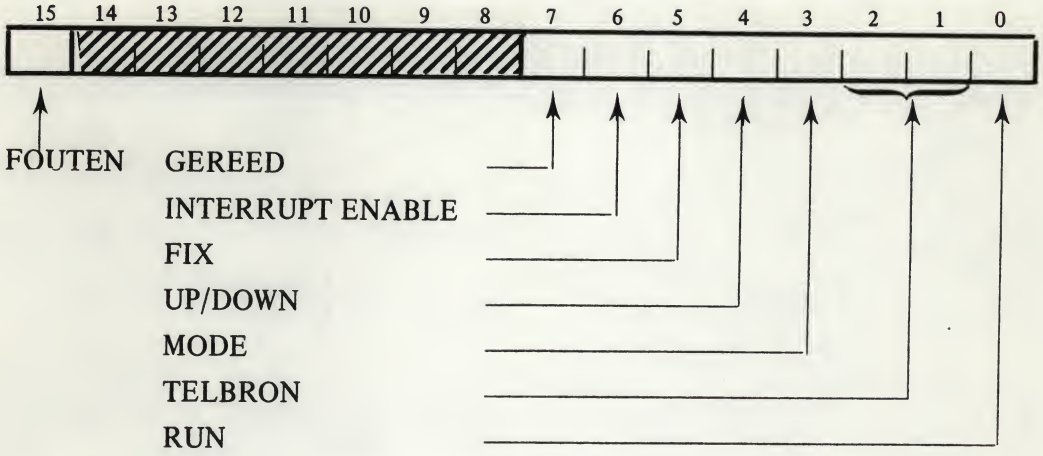


fig. 13. Indeling Control en Status Register

Verklaring Control/Status Register

Bit	Verklaring
15	Foutmeling ontstaat als een tweede interrupt ontstaat alvorens de eerste afgewerkt is.
14–8	Ongebruikt
7	Gereedmelding, na vol of leeg telling.
6	Interrupt Enable; als bit 6 hoog is ontstaat bij gereed een interrupt naar de locatie aangegeven in adres 104.
5	Fix-maintenance bit
4	Up/Down; bepaalt in welke richting geteld wordt, resp. 0 voor leegtellen en 1 voor voltellen.
3	Mode; Geeft aan of een ingesteld interval eenmaal of meerdere keren afgeteld wordt. Voor herhaald aftellen moet mode bit geset worden.
2,1	Telbron; hiervoor geldt volgend lijstje:

bit 2	bit 1	telbron
0	0	100khz
0	1	10 kHz
1	0	netfrekwentie
1	1	externe telbron

0	Run; bepaalt of de klok loopt. Run wordt in mode 0 gecleared door het gereed signaal.

Algemeen:

De tijd wordt bepaald door middel van een teller, die met verschillende in te stellen snelheden op of aftelt. Wanneer in de teller een overflow (resp. underflow bij aftellen) ontstaat, wordt het gereed bit (bit 7) in het Control/Status Register geset en wordt er, indien bit 6 (Interrupt Enable) hoog is een interrupt gegenereerd.
Staat de clock in mode 0, dan moet er een getal in de Counter Buffer geplaatst

worden. Bij het starten van de teller, d.m.v. het hoogmaken van het Run bit, wordt de inhoud van de buffer eerst in de teller geplaatst waarna het op- en aftellen begint. Bij een overflow (resp. underflow) wordt de inhoud van zowel Counter als Counter-buffer gecleared.
Staat de Clock in mode 1, dan moet er eveneens een getal in de buffer geplaatst worden, dat bij het hoogmaken van het Run-bit overgenomen wordt door de Counter. Bij elke overflow (resp. underflow) van de teller wordt nu echter de inhoud van de Counter Buffer opnieuw in de counter geplaatst en begint het op- of aftellen opnieuw.

Programma voorbeeld:

```
                    .=104
                    NAAM                ; Vullen Interrupt Vektor
                      340
CSR      = 172540
CSBR     = 172542                       ; Declaraties
COUNT    = 172544
                    .=2000              ; Start adres
                    MOV#100000.,CSBR    ; Instellen Counter beginwaarde
                    MOV#113,CSR         ; Start opdracht teller – Mode 1=aftellen.
                                        ; Telbron – 10kHz
                                        ; Interrupt Enable
START:              WAIT
                    BR START
NAAM:               .                   ; Interrupt Routine
                    .
                    .
                    RTI
```

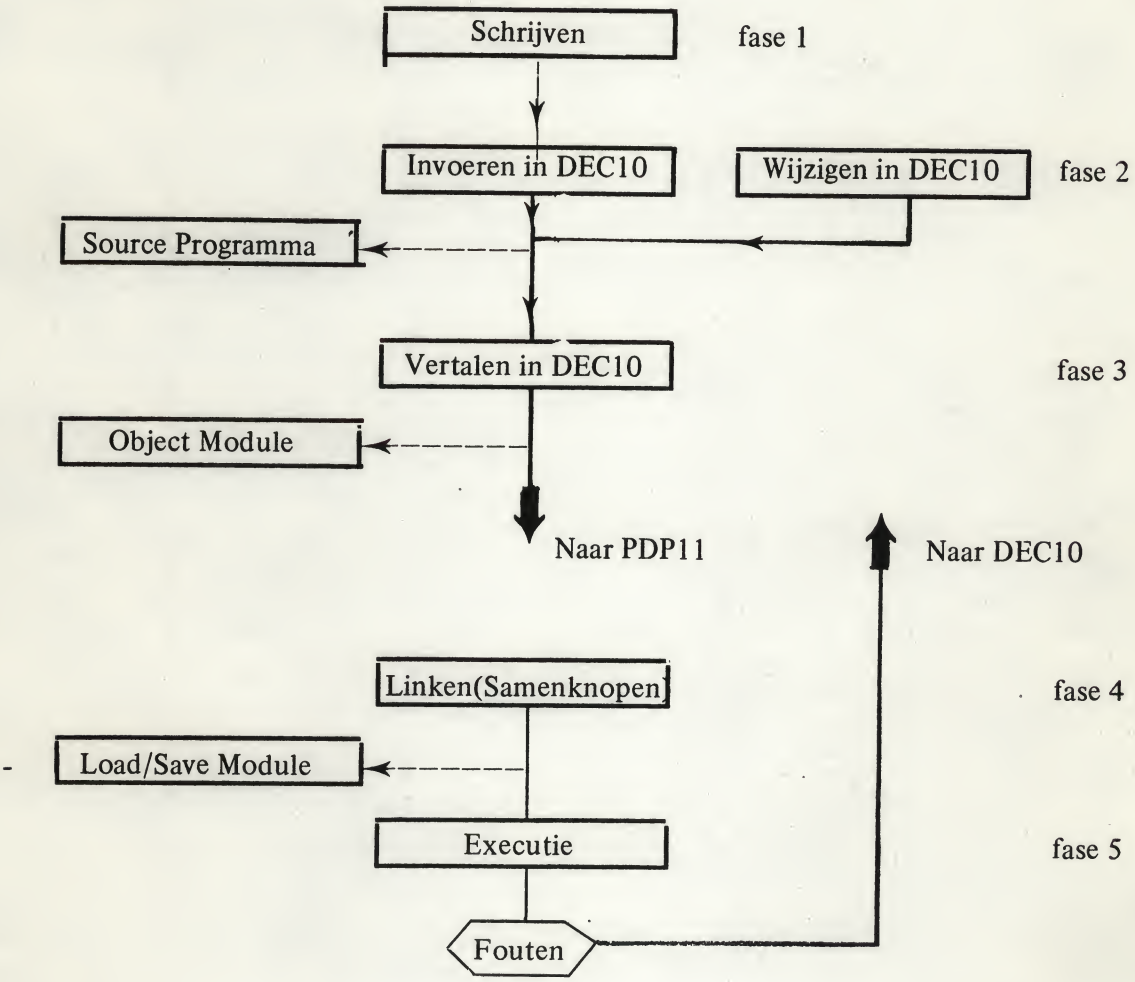
Toelichting:

De interrupt vektor is gevuld met het adres NAAM en het priotiteitsniveau 7.
Ontstaat er een interrupt dan springt het programma naar NAAM.
De telperiode is $10.000 \times \frac{1}{10.000} = 1$ sec.

HOOFDSTUK 2: PROGRAMMERING PDP11 COMPUTERS m.b.v. DE DEC 10

2.0. Inleiding

Bij de beschrijving van de gang van zaken bij het maken van een PDP11 programma wordt gebruik gemaakt van onderstaand blokdiagram. Schematisch is de gang van zaken weergegeven in fig. 14 en fig. 15.



De in het blokschema aangegeven fasen worden na elkaar beschreven.

Note: In de PDP11 computers wordt gewerkt onder het operating systeem RT11 versie 2 (single user)

2.1. fase 1 : Het schrijven van een PDP11 programma

De hier beschreven manier van werken geldt uitsluitend voor het programmeren in een assembler taal. (De hogere talen zoals BASIC en FORTRAN moeten aan de PDP11 verwerkt worden).
De assembler taal die gebruikt moet worden is MACRO 11.
Aangezien de taal MACRO 11 een uitbreiding is van de taal PAL 11, mag deze taal ook gebruikt worden.

Naast zelf te realiseren macro's, is het ook mogelijk gebruik te maken van standaard macro's. Bij een operating systeem zoals RT11 wordt een set standaard macro's, behorend bij dat operating systeem, geleverd. Bij RT11 worden deze macro's Programmed Request's genoemd. In hoofdstuk 3 wordt een samenvatting van de MACRO 11 taal gegeven.

2.2. fase 2 : Het invoeren resp. wijzigen van het (source) programma

Het invoeren resp. wijzigen van het programma kan op de volgende manieren gebeuren:

1° In de vorm van ponskaarten voor verwerking in de BATCH of via een REMOTE JOB ENTRY station.

2° In de vorm van een file op schijf m.b.v. een time sharing terminal.

Het programma, dat na het invoeren in de DEC 10 aanwezig is, wordt het SOURCE programma genoemd. Het is in ASCI I code aanwezig.
Het wijzigen van programma's is bij de eerste methode eenvoudig. Dit gebeurt door het verwijderen en tussenvoegen van ponskaarten.
Wordt gewerkt met een time sharing terminal, dan moeten verbeteringen in het SOURCE programma aangebracht worden m.b.v. het tekst-editing programma TECO. Hiervoor moet de gebruiker bekend zijn met de TECO commando's.
De gegevens van het TECO programma zijn verkrijgbaar bij het rekencentrum.

2.3. fase 3 : Het vertalen van het PDP11 programma

In de DEC10 computer is een vertaalprogramma MACY 11 aanwezig, waarmee het mogelijk is PDP11 source programma's te vertalen naar PDP11 object modules. Object modules zijn vertaalde programma's die nog niet aan een bepaalde plaats in het geheugen gebonden zijn en waarin nog "relaties" met andere programma's voor kunnen komen, die nog niet "ingevuld" zijn.
Aan het MACY11 programma moet meegedeeld worden welke input files samen vertaald moeten worden en waar (en/of onder welke naam) de outputs van de vertaling zijnde object module en listing, geplaatst moeten worden.
Wordt gebruik gemaakt van de z.g. programmed requests, dan moet in ieder geval de standaard macro file (SYSMAC. SML) meevertaald worden.
Het vertaalde programma, de object module, moet nu naar de PDP11 gebracht worden. Dit gaat als volgt:
M.b.v. de DEC 10 wordt een ponsband gemaakt, die m.b.v. het programma LOAD in de PPD11/10 op schijf wordt geplaatst.

2.4. fase 4 : Het linken of samenknopen

De object module, die tijdens de vertaling met MACY11 gemaakt is, **moet** gelinked worden. Dit linken kan plaats vinden samen met reeds vertaalde programmadelen of standaardprogramma's *of* met niets (als zichzelf).
Het linken vindt plaats op een PDP11 computer, bij voorkeur op de PDP11E10 (met schijfgeheugen), en zeker op de PDP11E10 wanneer van standaard programma's gebruik wordt gemaakt.

De standaardprogramma's zijn opgenomen in een standaard bibliotheek, die zodanig is opgebouwd, dat tijdens het linken alleen die programmadelen worden bijgevoegd die ook echt worden gebruikt. Er ontstaat dus geen programma overhead. Na het linken ontstaat, naar wens, een SAVE module of een LOAD module. Een SAVE module kan rechtstreeks met een RUN commando worden geëxecuteerd. Een LOAD module kan in elke PDP11 computer (ook zonder operating systeem) worden ingelezen m.b.v. een ABSOLUTE LOADER.

2.5. fase 5 : De executie van een PDP11 programma

De executie van een SAVE module is, zoals reeds bij fase 4 aangegeven, erg eenvoudig en gebeurt met een RUN commando.
Om fouten te zoeken in een programma kan het voordeel hebben te beschikken over een z.g. ODT programma. (ODT = On line Debugging Technique.)
Dit ODT-programma moet in fase 4 samengeknoopt worden met het gebruikersprogramma. De executie van het programma onder ODT vindt nu plaats door:

1° Met een GET commando het programma inclusief ODT in het werkgeheugen te halen.
2° Met een START commando te starten op het startadres van ODT, dat te vinden is in de linkmap onder de naam O.ODT.

Opmerkingen:
Op de standaard programma's wordt ingegaan in hoofdstuk 4.
Het LINK programma en het ODT programma worden behandeld in hoofdstuk 5.
Het RUN commando wordt eveneens behandeld in hoofdstuk 5.

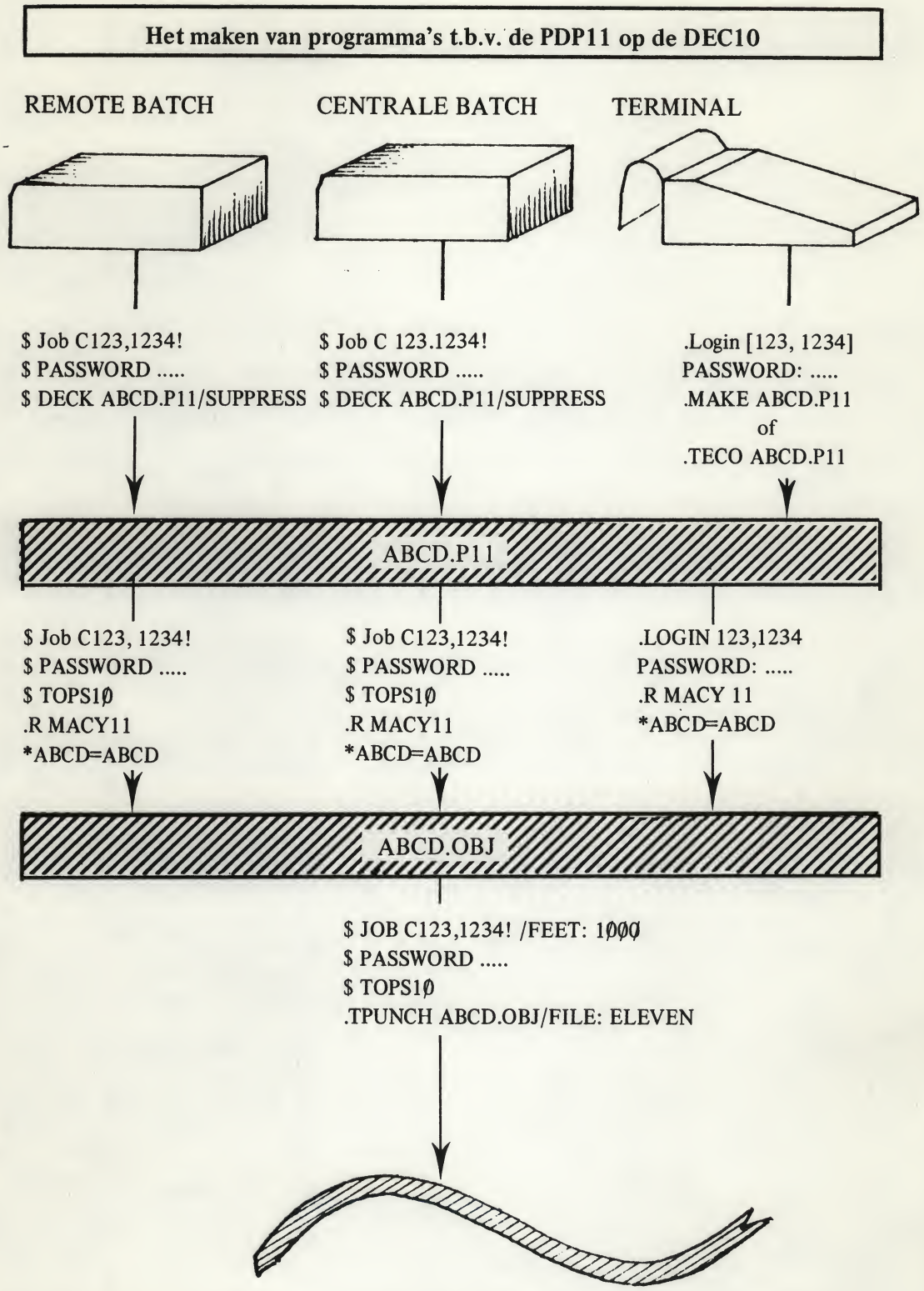


fig. 14: Schema van de handelingen om m.b.v. DEC10 een PDP11 programma te vertalen.

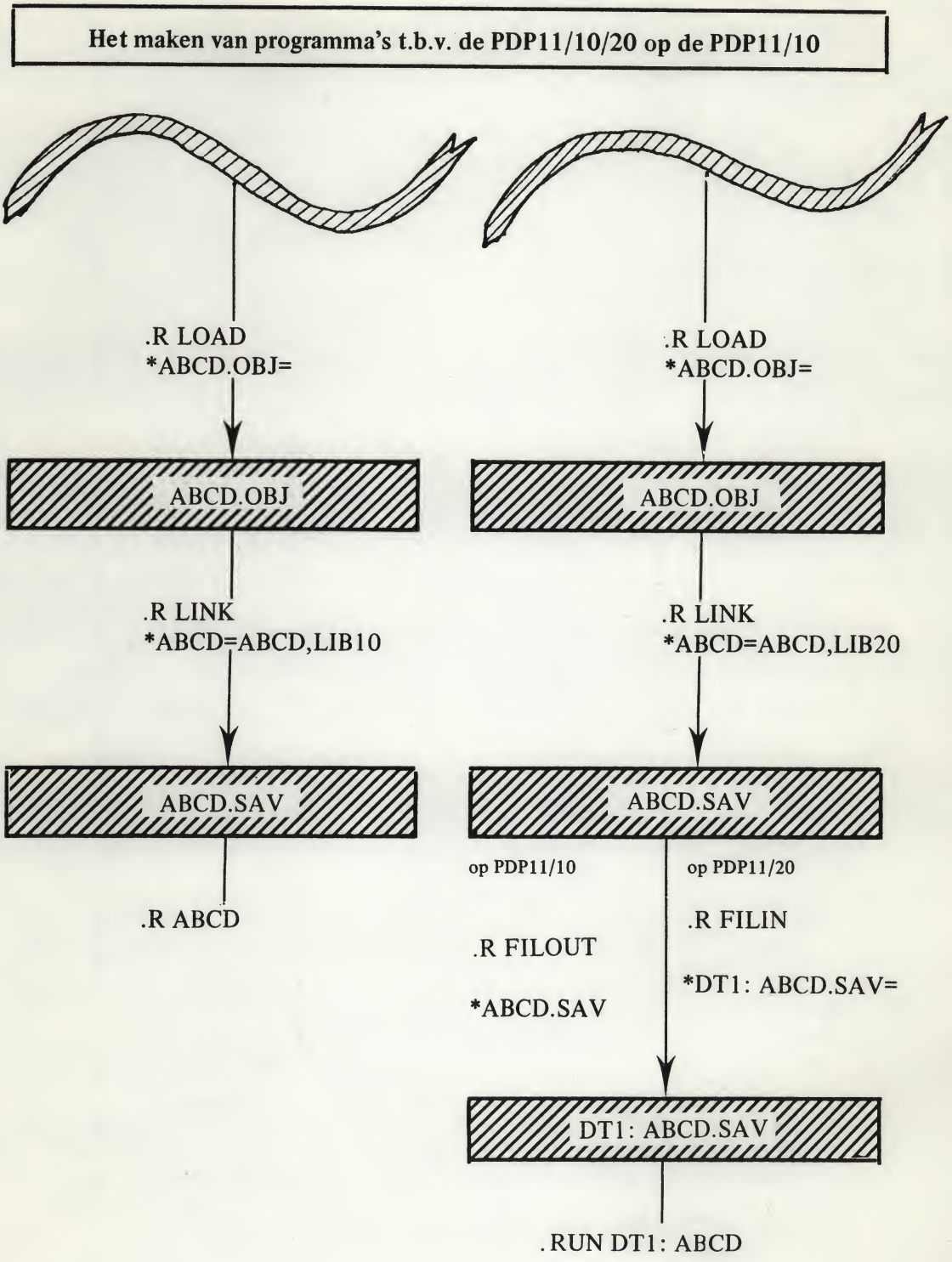


fig. 15: Schema van de gang van zaken bij het linken van een PDP11 programma, dat op de DEC10 vertaald is.

HOOFDSTUK 3: ASSEMBLER PROGRAMMERING PDP11

3.0. Inleiding

De PDP11 kan geprogrammeerd worden in de assemblertaal PAL11. Een uitgebreidere taal, waarin ook gebruik gemaakt kan worden van macro's is MACRO11. Beide talen zijn toegestaan, mits gebruik wordt gemaakt van een MACRO11 vertaalprogramma.
Op de DEC10 is dit het programma MACY11, op de PDP11 is dit het programma MACRO.
De assembleertaal maakt het mogelijk een voor de gebruiker begrijpelijke symbolisch "source" programma te schrijven, waarbij gebruik kan worden gemaakt van letter, cijfers, symbolen en symbolisch adresseren.

3.1.0. Statements-Instructie regels

Een programma wordt opgebouwd uit instructieregels, ook wel statements genoemd. Een statement moet de volgende gedaante hebben:

Label: operator operand(en) ; comment

De statements worden gescheiden door een (RETURN) of (CR) karakter. De label- en commentvelden behoeven niet gevuld te zijn.

3.1.1. Gegevens label

Een label is een door de gebruiker te definieren symbolische naam (adres). Het is een referentie aan een plaats in een programma. Een label mag bestaan uit letters, cijfers, dollartekens. Andere karakters zijn fout en resulteren in een foutmelding.
Voor een label gelden verder de volgende regels:

1. Het eerste karakter mag geen cijfer zijn.
2. Een label mag niet uit meer dan zes karakters bestaan.
3. Er mogen geen dezelfde labels gebruikt worden.
4. Een label mag geen spaties of tabulaties bevatten.
5. Een label *moet* afgesloten worden met een dubbele punt (kolom).

Wanneer een label uit meer dan zes karakters bestaat, dan zijn de eerste zes karakters bepalend.
Een foutcode M ontstaat als twee dezelfde (in de eerste zes karakters) labels zijn gebruikt.
De foutcode I ontstaat wanneer in een label een niet toegestaan karakter wordt gebruikt. Het is toegestaan aan een instructieregel meerdere labels toe te kennen. Deze labels moeten alle met een dubbele punt afgesloten zijn. Aan alle labels wordt in dit geval dezelfde waarde (adres) toegekend.
Voorbeeld statement met meerdere labels:

```
NAAM:   DAG:   UUR:        MOV   A,B
```

3.1.2. Gegevens operator

De operator kan een operatie mnemonic of een assembler besturingscode zijn. Het is altijd een code waarvan de betekenis vastligt.
De operator volgt op een label (indien aanwezig). Het operator gedeelte van een statement wordt meestal afgesloten door een spatie. Een van de volgende karakters is echter ook toegestaan:

#	+	@	(	”	’
%	-	!	)	&	; ,

De betekenis van deze karakters volgt.

De mnemonische operatie codes zijn te vinden in hoofdstuk 1 en in appendix B.

3.1.3. Gegevens operanden

De operanden van een instructie geven de locaties waarop de bewerking plaats heeft. Een operand kan zijn: *een symbool, een adresexpressie of een getal.*

Wanneer in een instructie meerdere operanden voorkomen, moeten ze door komma's gescheiden worden. Het operandengedeelte wordt afgesloten door een puntkomma (semicolon) als kommentaar volgt, anders door een (RETURN)- of (CR) karakter.

Voorbeeld:

Label: MOV A,(Ro) ; Comment

3.1.4. Gegevens comment

In het comment gedeelte mag kommentaar gegeven worden. Hierin mogen alle karakters gebruikt worden, met uitzondering van:

RUBOUT, RETURN, CNTL/FORM.

Het kommentaar gedeelte moet voorafgegaan worden door een puntkomma (semicolon).

3.1.5. Overzicht in de listing

Om een overzichtelijke listing te verkrijgen, is het toegestaan tussen de verschillende delen van een instructieregel, spaties of tabulaties te gebruiken. Deze hebben geen invloed op de assemblage.

3.2.0. Symbolen

Er zijn twee soorten symbolen:

1° Gereserveerde symbolen.

2° Door de gebruiker gedefinieerde symbolen.

Gereserveerde symbolen zijn de mnemonische operatie codes en de assembler besturingscodes. Deze codes zijn vast opgenomen in een assembler symbolen tabel.

Gedurende de assemblage wordt deze tabel met gebruikerssymbolen uitgebreid. Gebruikt de gebruiker eenzelfde symbool als een gereserveerd symbool, dan wordt gedurende deze assemblage het door de gebruiker gedefinieerde symbool gebruikt en het gereserveerde symbool genegeerd. Voor de gebruikerssymbolen gelden dezelfde regels als voor de *labels.*

3.3.0. Assignment Statements

Assignment statements zijn opdrachten aan de assembler om aan een symbool een waarde toe te kennen of om een symbool met een waarde te associeren. De vorm van een assignment statement is de volgende:

Symbool = expressie

Voorbeelden:

```
Ro      =   %o
TPB     =   177566
NAAM    =   100 + PLAATS
PLAATS  =   200
```

Bij de assignment statements moeten de volgende regels in acht worden genomen:

1. Het symbool en de expressie moeten door een =teken worden gescheiden.
2. Een assignment opdracht mag voorafgegaan worden door een label en gevolgd worden door kommentaar.
3. Er kan niet meer dan een symbool per statement gedefinieerd worden.
4. Er mag niet meer dan 1 stap voorwaarts gerefereerd worden.

   Voorbeeld:

   ```
   X   =   Y
   Y   =   Z
   Z   =   1
   ```

   Dit voorbeeld levert een 2 staps voorwaarts referentie en is dus fout.

Een reeds eerder gedefinieerd symbool kan opnieuw gedefinieerd worden. De laatste definitie blijft dan voor het gehele programma van kracht.

Voorbeeld:

```
PLAATS:     A   =   1
NAAM:       MOV #   A, MASK
```

Betekenis: Het symbool A krijgt de waarde 1 toegekend, de symbolen PLAATS en NAAM krijgen waardes toegekend die voortvloeien uit de programma positie. De waarde van PLAATS is gelijk aan die van NAAM, omdat het assignment statement A = 1 geen machinecode oplevert.

3.3.1. Algemene register symbolen

De acht algemene registers van de PDP11 zijn genummerd van 0 t/m 7. De binnen het assembler programma bekend zijnde symbolen voor de algemene registers zijn de nummers 0 t/m 7 voorafgegaan door een % (procent) teken.

Voorbeelden:

```
R0  = %0        ; Register 0 als R0
R1  = %1        ; Register 1 als R1
TEL = %3        ; Register 3 als TEL
SP  = %6        ; Register 6 als Stack pointer
PC  = %7        ; Register 7 als Program counter
```

Na deze declaraties, zoals in de voorbeelden gegeven, kan in het programma van deze symbolen gebruik gemaakt worden om registers aan te duiden. Natuurlijk mogen ook de bij de assembler bekende symbolen %0 t/m %7 gebruikt worden.

Voorbeelden:

```
CLR     R0      ; mits R0 = %0 gedefinieerd is
CLR     %0
```

3.4.0 Expressies (rekenkundige uitdrukkingen)

Symbolen, getallen en ASCII karakters kunnen gebruikt worden om door middel van rekenkundige of logische operatoren expressies te vormen. Ook mag de momentane waarde van de assembler locatie teller hierin gebruikt worden. De expressies worden van links naar rechts berekend, waarbij niet op haakjes wordt gelet.
Een ontbrekende term wordt 0, een ontbrekende operator wordt +.

Voorbeeld:

A + - 100 wordt : A + 0 - 100
TAK! LA 17777 wordt: TAK! LA + 17777

Bij een ontbrekende operator ontstaat foutmelding Q ondanks het voortzetten van de berekening. Als de expressie voor een byte locatie bedoeld is, wordt deze eerst berekend op woord formaat, waarna afbreking op 8 bits plaats heeft.

3.4.1. Getallen

In expressies worden octale en decimale cijfers geaccepteerd. Decimale cijfers moeten voorzien zijn van een decimale punt. Wanneer in een getal een cijfer 8 of 9 voorkomt, zonder dat er een decimale punt aanwezig is, ontstaat foutmelding N, en wordt het getal als een decimaal getal gebruikt. Negatieve getallen moeten voorafgegaan worden door een –teken, positieve getallen mogen voorafgegaan worden door een +teken. Wanneer een getal te groot wordt voor een computerwoord of byte, dan wordt het van links af ingekort en ontstaat foutmelding T.

3.4.2. Rekenkundige en logische operatoren

In expressie mogen enige rekenkundige en logische operatoren gebruikt worden. De toegestane rekenkundige operatoren zijn:

+	plus	geeft aan of optellen of een positief getal
–	min	geeft aan of aftrekken of een negatief getal.

De toegestane logische operatoren zijn:

&	AND	geeft een logische AND aan
!	OR	geeft een logische OR aan

3.4.3. ASCII karakters

De bij de PDP11 computers werken de terminals met karaktercodes volgens de ANSCII norm. Deze code heet de ASCII code. Karakters die voor de terminal bedoeld zijn en karakters die met behulp van de terminal ingevoerd worden zijn dus ASCII gecodeerd. De ASCII code bestaat uit 7 bits. Bij invoer wordt deze 7 bits code door een 1-bit voorafgegaan om juist een byte te kunnen vullen.
Wanneer in een expressie een karakter als een ASCII code verwerkt moet worden, moet dit karakter door een quote voorafgegaan worden.
Moeten twee karakters tezamen als ASCII codes in één computerwoord geplaatst worden, dan moeten deze twee karakters door één "dubble quote" voorafgegaan worden.

Voorbeelden:

'A ⟶ 101_8 ⟶ 01 000 001
"AB ⟶ 102_8 | 101_8 ⟶ 0 100 001 0 | 01 000 001
high order byte low order byte

3.4.4. Assembler locatie teller

Gedurende de assemblage wordt bijgehouden op welke plaats een vertaalde machinecode terecht gaat komen. Dit gebeurt m.b.v. de assembler locatie teller. Van deze teller kan in een programma gebruik worden gemaakt. Het symbool voor de assembler locatie teller is de punt (.).
Als het gebruikt wordt in het operanden veld van een instructie, vertegenwoordigt de punt het adres van het eerste woord van de instructie.
Aan het begin van elke assemblage wordt de assembler locatie teller nul gemaakt. Daarna worden, normaal gesproken, op een volgende locaties aan de te genereren machinecode toegekend. Deze gang van zaken kan verstoord worden door in de programmatekst aan de assembler locatie teller een nieuwe waarde toe te kennen. Dit gaat als volgt:

```
. = expressie
```

Voorbeelden:

```
. = 1000          ; teller wordt 1000
. = . + 100       ; teller wordt verhoogd met 100
```

De expressie mag geen voorwaartse referenties bevatten en geen symbolen, die gedurende verschillende doorgangen verschillende waarden hebben.

Voorbeeld:

```
        . = 500
FIRST:  MOV  .+10, COUNT
```

Betekenis : De locatie teller wordt gevuld met 500. Hierdoor krijgt de label FIRST de waarde 500 toegekend. De uitdrukking .+10 levert in dit geval 510, zodat met behulp van de MOV instructie de inhoud van locatie 510 in de locatie COUNT geplaatst wordt.

Reservering geheugenruimte

Geheugenruimte kan gereserveerd worden door het volgende statement te programmeren:

```
BUFFER :  . = . + 100.
```

Dit resulteert in het overslaan, dus reserveren, van 100 byte posities vanaf de locatie BUFFER.
De mode van de assembler locatie teller kan beinvloed worden met de assembler besturingscodes .ASECT en .CSECT. Hierover volgt later meer bij de bespreking van de assembler besturingscodes.

3.5. Adressering

In het operanden gedeelte van een instructie kunnen allerlei expressies gebruikt worden, die alle uiteindelijk moeten resulteren in adresseervelden, bestaande uit een adresseermode en een registernummer zoals aangegeven is in hoofdstuk 1.
De programmateller bevat tijdens de executie van een instructie altijd het adres van het volgende geheugenwoord.
Hiervoor geldt de volgende regel:
Telkens wanneer de programmateller wordt gebruikt om iets uit het geheugen te halen, wordt deze met twee opgehoogd.
Wanneer een source operand het volgende woord in het geheugen gebruikt als adres of dataruimte, wordt de programmateller gebruikt om dit woord uit het geheugen te halen en dus tevens met twee opgehoogd.

Voor de destination operand geldt hetzelfde. Het is dus mogelijk, dat de programmateller tijdens het executeren van een instructie 6 byte posities verder gaat wijzen dan de locatie van de eigenlijke instructie, als zowel de source operand als de destination operand van een extra geheugenplaats gebruik maken.

3.5.1. Adresseermodes

De adresseermodes zijn reeds in hoofdstuk 1 besproken. Hier wordt slechts een aanvulling gegeven. Bij het beschrijven van de adresseermodes gelden de volgende conventies:

a : E betekent een expressie zoals beschreven in 3.4.0.

b : R geldt als een register expressie. Dit zijn expressies die beginnen met een procent (%) teken of symbolen, die als registers gedefinieerd zijn.

Voorbeelden:
```
R0 = %0      ; Register 0
R1 = R0+1    ; Register 1
R2 = 1+R1    ; Register 2
```

c : ER geldt voor een register expressie of een willekeurige expressie die 0 t/m 7 oplevert.

d : A geldt als een algemeen symbolische adresaanduiding.

De verschillende adresseermethodes worden beschreven met bovengenoemde expressies. Hierbij wordt verder gebruik gemaakt van de reeds in hoofdstuk 1 besproken instructies MOV en CLR. Ook voor de overige instructies wordt naar hoofdstuk 1 verwezen.

3.5.1.1. Register Mode

Algemene vorm: OPR R
Betekenis: Register R is de operand
Voorbeeld:
```
CLR R0          ; R0: = 0
MOV R0, R1      ; R1: = R0
```

3.5.1.2. Deferred Register mode

Algemene vorm: OPR @R of OPR (R)
Betekenis: Register R bevat het adres van de operand.
Voorbeeld:
```
CLR (R0)        ; (R0): = 0
MOV R0, @ R1    ; (R1): = R0
```
Stel:
inhoud R0 is 100
inhoud R1 is 200
dan: adres 100 wordt 0
adres 200 wordt 100

3.5.1.3. Auto increment mode

Algemene vorm: OPR (ER) +

De inhoud van het register gespecificeerd door ER wordt, na te zijn gebruikt als adres van de operand, opgehoogd met één voor een byte instructie en met twee voor een woordinstructie.

Voorbeeld: CLR (R1)+
Uitwerking,
Stel:
inhoud R1 = 100
dan wordt adres 100 gelijk 0,
daarna wordt R1 102 gemaakt.

3R *3.5.1.4. Deferred auto increment mode*

Algemene vorm: OPR @ (ER)+

De inhoud van het register gespecificeerd door ER wijst naar het adres van de operand. De inhoud van het register wordt, na als adreswijzer te zijn gebruikt, opgehoogd.

Voorbeeld: CLRB @ (R2)+

Uitwerking,

Stel: R2 = 2000
en de inhoud van 2000 is 1000
dan wordt byte locatie 1000 nul gemaakt.
Hierna wordt R2 2001 gemaakt.

4R *3.5.1.5. Auto decrement mode*

Algemene vorm: OPR –(ER)

Alvorens de inhoud van het register, dat door ER gespecificeerd wordt, gebruikt wordt als adres van de operand, wordt deze verlaagd.

Voorbeeld: CLR –(R5)

Uitwerking,

Stel: inhoud R5 = 1000
dan wordt eerst R5 met twee verlaagd, dus 776 gemaakt.
Daarna wordt de nu door R5 aangewezen locatie 776 nul gemaakt.

5R *3.5.1.6. Deferred auto decrement mode*

Algemene vorm: OPR @ – (ER)

Alvorens de inhoud van het register gespecificeerd door ER wordt gebruikt als aanwijzer van het adres van de operand, wordt deze verlaagd.

Voorbeeld: CLR @ – (R1)

Uitwerking,

Stel: inhoud R1 is 1000
de inhoud van 1000 is 500
de inhoud van 776 is 300
dan wordt eerst R1 met twee verlaagd tot 776,
daarna wordt de locatie aangewezen door 776 nul gemaakt, dus locatie 300 wordt nul.

6R *3.5.1.7. Index mode*

Algemene vorm: OPR E (ER)

Het adres van de operand wordt bepaald door de inhoud van het register aangeduid met ER, vermeerdert met de expressie E.

Voorbeeld: CLR B 5(R2)

Uitwerking,

Stel: de inhoud van R2 is 1000.
dan wordt byte locatie 1005 nul gemaakt.

7R *3.5.1.8. Indexed deferred mode*

Algemene vorm: OPR @ E (ER)

Het adres van de operand wordt aangewezen door de inhoud van het register, aangeduid met ER, vermeerdert met de expressie E.

Voorbeeld: CLR @ 500 (R2)
Uitwerking,
Stel: inhoud R2 is 500,
dan wordt het adres van de locatie die nul gemaakt moet worden, gevonden in locatie 1000.
Is de inhoud van 1000 b.v. 2000, dan wordt locatie 2000 nul gemaakt.

3.5.1.9. Direkte operand

Algemene vorm: OPR # E

De expressie E wordt als direkte operand, dus b.v. als getal gebruikt. De adresseermode maakt gebruik van de momentane stand van de programmateller. De waarde van de expressie E verschijnt in absolutie vorm in de locatie volgend op de instructie regel.

Voorbeeld: MOV # 1000, R3

Uitwerking: Het getal 1000 (in octale betekenis) wordt in Register 3 geplaatst.

3.5.1.10. Absoluut adres

Algemene vorm: OPR @ # E

De expressie E wordt als absoluut adres gebruikt. Deze mode kan met vrucht worden gebruikt, wanneer een programma wordt verplaatst. De waarde van de expressie E wordt weer in de locatie volgend op de instructieregel geplaatst.

Voorbeeld: CLR @ # 10

Uitwerking: Onafhankelijk van de verplaatsing van het programma in het geheugen, wordt locatie 10 nul gemaakt.

3.5.1.11. Relatief adres

Algemene vorm: OPR E

Het gebruik van symbolische adressen levert de relatieve mode. Op de locatie volgend op de instructieregel komt het verschil tussen de momentane stand van de programmateller en de met E bedoelde operand.

Voorbeeld: MOV A,B

Uitwerking: De inhoud van symbolisch locatie A wordt geplaatst in de symbolische locatie B.

3.5.1.12. Deferred relatieve adres

Algemene vorm: OPR @ E

In dit geval wordt een symbolisch adres of een andere uitdrukking voor E gebruikt als aanwijzen voor de operand. Op de locatie volgend op de instructieregel komt het verschil tussen de momentane stand van de programmateller en de waarde van de expressie E.

Voorbeeld: CLR @ A

Uitwerking: Indien het symbolisch adres A de waarde 1000 bevat, wordt locatie 1000 nul gemaakt.

3.6. Instructies

De instructies van de PDP11 met hun mnemonische operatiecodes zijn reeds in hoofdstuk 1 beschreven. Hier worden slechts de verschillende vormen van instructies gegeven. Zij zijn weergegeven in onderstaande tabel.

Soort	Vorm	Voorbeeld
Dubbel Operand	OPR A,A	MOV (R6)+, NAAM
Enkel Operand	OPR A	CLR –(R2)
Operaties	OPR	HALT
Sprong	OPR E	BR NAAM + 2
Subroutine sprong	JSR ER,A	JSR PC, SUBR
Subroutine return	RTS ER	RTS PC
EMT/TRAP	OPR	EMT
	of OPR E	EMT 31

Opmerking: Bij de sprongopdrachten mag de sprongafstand niet groter zijn dan 128 plaatsen achterwaarts en 127 plaatsen voorwaarts. Bij grotere sprongafstanden moet van de JMP instructie gebruik worden gemaakt.

3.7. Assembler besturingscodes

Assembler besturingscodes (assembler directives) zijn commando's aan het assembler vertaalprogramma. Ze mogen voorafgegaan worden door een label en gevolgd worden door kommentaar. Een assembler besturingscode staat in het operatorgedeelte van een instructie.

3.7.1. .TITLE

De .TITLE assembler besturingscode wordt gebruikt om aan de na de vertaling ontstane object module een naam te geven. Wanneer geen .TITLE gespecificeerd wordt, wordt aan de object module de naam .MAIN. toegekend.

Voorbeeld : .TITLE NAAM 1

3.7.2. .GLOBL

De .GLOBL assembler besturingscode wordt gebruikt om symbolen als globalen te definieren. Globalen zijn symbolen die of in een ander afzonderlijk te vertalen programma (b.v. subroutine) gedefinieerd zijn, of binnen dit programma gedefinieerd zijn maar in een ander afzonderlijk te vertalen programma gebruikt (kunnen) worden. De vorm van de .GLOBL besturingsinstructie is:

.GLOBL NAAMA, NAAMB, NAAMC ....., NAAMN

Bij fouten in een .GLOBL statement, wordt veelal zonder foutmelding het gedeelte na de fout genegeerd.

3.7.3. Programma sectie codes (.ASECT en .CSECT)

Het assembler (Macro) vertaalprogramma levert reloceerbare objectmodules. Dit zijn programmadelen die in de samenknoopfase samengevoegd worden en dan pas aan een geheugenpositie gebonden. Veelal wordt in deze fase een aantal programmadelen aaneengesloten achter elkaar "geplakt". Het kan echter ook voorkomen, dat het gewenst is, programmadelen aan vaste geheugenposities te binden. Is dit het geval, dan moet dan aangegeven worden door aan het begin van het programmadeel een .ASECT besturingscode op te nemen.

Om het programma dan op een gewenste plaats te krijgen, moet daarnaast de assembler locatie teller een waarde gegeven worden. Voor een programmadeel, dat op adres 1000 beginnen moet, ziet dit er als volgt uit:

```
.ASECT
.=1000
```

Wanneer programmadelen niet aan een vaste plaats gebonden zijn, mag de .CSECT besturingscode worden gebruikt. Wordt noch de .ASECT, noch de .CSECT code gebruikt, dan wordt het programma vertaald alsof er een .CSECT is gegeven.

3.7.4. Assembler besturingscodes voor het plaatsen van data

Een aantal soorten gegevens kunnen door het assembler vertaalprogramma in programma locaties geplaatst worden, zoals:

ASCII karakters in byte posities
Integer getallen in woordposities
Floating point getallen in 2woorden posities.

3.7.4.1. .BYTE

Met de .BYTE code kan een byte locatie gevuld worden. De .BYTE code mag gevolg worden door meerdere, door komma's gescheiden expressies. De waardes van deze expressies worden alle in opeenvolgende byte locaties geplaatst. De vorm van een .BYTE statement is als volgt:

```
.BYTE  exp1, exp2, .....expn
```

Past de waarde van een expressie niet in een byte, dan wordt deze afgebroken en ontstaat foutmelding T.

Voorbeeld : Tekstbuffer met de codes 15 en 12, die een carriage return en een linefeed voorstellen.

```
TEBUF:  .BYTE 15,12
```

3.7.4.2. .WORD

Met de .WORD code kunnen woordlocaties gevuld worden. De .WORD code mag gevolgd worden door meerdere, door komma's gescheiden expressies. De waardes van deze expressies worden in opeenvolgende woordlocaties geplaatst.
De vorm van het .WORD statement is:

```
.WORD    exp1, exp2, .....expn.
```

Voorbeeld:

```
.=1420
.WORD  17500, .+4,  SAL
```

Betekenis: De assembler locatie teller is gevuld met 1420. Tengevolge van de .WORD instructie worden de geheugenplaatsen 1420 e.v. als volgt gevuld:

```
1420    17500
1422     1426
1424    de waarde toegekend aan SAL.
```

Ontbrekende operanden, waarvoor wel komma's aanwezig zijn, worden als nul geinterpreteerd. Een leeg operatorveld in een instructie wordt, als het gevolgd wordt door een of meer operanden, als een .WORD instructie geinterpreteerd.

3.7.4.3. .ASCII

De .ASCII besturingscode kan gebruikt worden om een ASCII (tekst) karakterstring op opeenvolgende byte posities in het geheugen te plaatsen. De karakters die niet in deze tekst opgenomen mogen worden zijn:

null, RUBOUT, RETURN, LINEFEED en FORMFEED.
De tekststring dient voorafgegaan en afgesloten te worden met een willekeurig, niet in de tekststring voorkomend karakter. Hiervoor mogen geen "puntkomma" of "is gelijk" teken gebruikt worden.
Het .ASCII statement is van de volgende vorm:

.ASCII /DIT IS TEKST/

Betekenis: **Op opeenvolgende locaties worden de ASCII codes voor DIT IS TEKST geplaatst.**

3.7.4.4. .ASCIZ

Een variant op het hiervoor behandelde .ASCII statement is het .ASCIZ statement. Hiervoor geldt vrijwel hetzelfde, de tekststring wordt alleen met een 0 byte aangevuld. Veel tekstverwerkende programma's gebruiken dit 0 byte als teken, dat de tekst afgelopen is.
Voorbeeld:

```
.=1000
.ASCIZ /NAAM/
```

Dit levert in het geheugen:

1001	A	N	1000
1003	M	A	1002
1005	.	0	1004
1007	.	.	1006
1011	.	.	1010

3.7.4.5. .RAD50

In PDP11 systeemprogramma's wordt soms in plaats van de ASCII code de compactere RADIX 50 code gebruikt. Bij de RADIX 50 code worden 3 karakters in een computerwoord geplaatst. Voor de rest geldt voor dit statement hetzelfde als voor het .ASCII statement.
De vorm van het .RAD 50 statement is:

.RAD 50 /ABC/

Wanneer minder dan drie karakters gebruikt zijn, wordt links met spaties aangevuld.
De toegestane karakters zijn:

De letters A t/m Z
De cijfers 0 t/m 9
De tekens $. en blank

3.7.5. Assembler locatie teller besturing

Zoals reeds in het voorgaande behandeld is, werkt het assembler vertaalprogramma met een locatieteller. Het .karakter is het symbool voor de locatieteller. Er bestaan verder een aantal assembler besturingscodes die de locatieteller beinvloeden. Deze worden hierna behandeld.

3.7.5.1. .EVEN

De .EVEN code zorgt er voor, dat de locatieteller, zo dat niet het geval is, even wordt gemaakt. Het .EVEN statement kent geen operanden.
De .EVEN code kan gebruikt worden om na een byte georiënteerde buffer ervoor te zorgen, dat de volgende instructie op een woordlocatie komt. Instructies mogen alleen op woord- dus even locaties voorkomen.

```
Voorbeeld:          TEBUF:   .ASCIZ    /NAAM/
                             .EVEN
                    DOOR:    MOV       AAP, TOUW
                               .
                               .
                               .
                               .
```

Betekenis: Na vertaling ontstaat een uit 5 bytes bestaande tekstbuffer, omdat de string NAAM met een nul byte wordt aangevuld. Om er voor te zorgen, dat de instructie met de label DOOR op een woordlocatie komt, is het .EVEN statement opgenomen.

3.7.5.2. .ODD

De .ODD code zorgt ervoor, dat de programmateller, zo dat niet het geval is, oneven wordt door er een bij te tellen.

3.7.5.3. .BLKB en .BLKW

De .BLKB en .BLKW codes dienen om een stuk geheugen te reserveren. De vorm van het statement is resp.:

```
.BLKB   exp
.BLKW   exp
```

Bij het .BLKB statement geeft de waarde van de expressie aan, hoeveel byte posities gereserveerd worden. Bij het .BLKW statement geldt hetzelfde, maar dan voor woordposities.

3.7.6. Direkte invoer van floating point getallen

Omdat vrijwel alleen met tweewoords floating point getallen wordt gewerkt, wordt daar alleen aandacht aan besteed. In fig. 16 is de woordenindeling van een tweewoords floating point getal weergegeven.

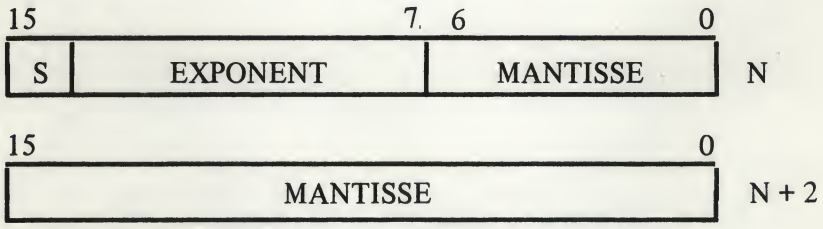


fig. 16 Indeling tweewoords floating point getal

De assembler besturingscodes voor het plaatsen van floating point getallen zijn resp.:

```
.FLT 2   arg1, .....,arg n     voor 2woords getallen
.FLT 4   arg1, .....,arg n     voor 4woords getallen
```

De argumenten arg1 t/m arg n moeten floating point getallen zijn in een van de volgende gedaantes:

```
3
3.0
3.
3.0 E 0
3 E 0
.3 E 1
300 E – 2
```

(Alle representaties van het getal 3)

Om de floating point getallen in het beperkte aantal bits van twee woorden weer te geven, worden de getallen afgerond. Wanneer afbreking de voorkeur verdient met de assembler besturingscode .ENABL FDT gebruikt worden. Hierna worden bij de vertaling alle floating point getallen afgebroken. Opnieuw terugkeren naar de afrond mode kan met de assembler besturingscode .DSABL FDT.

Voorbeeld: *Plaatsen floating point getallen:*

```
.=5000
.FLT 2          1.0, 2 E 1, 500
```

De locaties 5000 t/m 5014 worden achtereenvolgens gevuld met de floating point voorstellingen van de getallen 1, 20 en 500.

3.7.7. Scheidingsbesturingscodes

3.7.7.1. .END

De .END code geeft het eind aan van het te vertalen programma. De .END code mag slechts door één operand gevolgd worden en wel een expressie om het beginpunt van het programma aan te geven. Na het laden van het programma start het programma automatisch bij dit opgegeven beginpunt.

Voorbeeld:

```
.END    START
```

Na het laden start het programma automatisch op de locatie START.

3.7.7.2. .EOT

De .EOT code kan het einde aanduiden van een programmastuk, doch niet het totale einde van het programma. Bij het vertalen van bij elkaar horende ponsbanden, kan dit gebruikt worden. Onder het RT11 systeem wordt, omdat alleen files vanaf massamedium vertaald kunnen worden, het .EOT statement genegeerd.

3.7.7.3. .LIMIT

Wanneer het binnen een programma nodig is te weten tot hoever in het geheugen het programma loopt, dan kan een .LIMIT code opgenomen worden. Tengevolge van de .LIMIT code worden twee woorden gereserveerd waarin door het LINK programma het laagste en hoogste in gebruik zijnde geheugenadres wordt geplaatst.

3.7.8. Conditionele assemblages

Conditionele assembler besturingscodes maken het mogelijk naar wens bepaalde programmadelen wel of niet mee te vertalen. Hiervoor bestaan een aantal IF statements, die afhankelijk van de waarde van de er achter gespecificeerde grootheid, er voor zorgen, dat het volgende programmadeel wel of niet vertaald wordt. De toegestane statements zijn:

```
.IFZ      E     ; programmadeel wordt vertaald als E=0
.IFNZ     E     ;      „        E ≠ 0
.IFL      E     ;      „        E < 0
.IFLE     E     ;      „        E ≤ 0
.IFG      E     ;      „        E > 0
.IFGE     E     ;      „        E ≥ 0
.IFDF     Q     ; Q = 1
.IFNDF    Q     ; Q = 0
```

Hierin is E een expressie en Q een logische combinatie van symbolen, zoals
A ! B & C. (A of B en C).
Voorbeeld: .IFDF A!B
Als A of B als 1 gedefinieerd is, wordt het volgende programmadeel vertaald.
Het programmadeel waarop het voornoemde IF statement betrekking heeft, moet afgesloten worden met een .ENDC statement.
Voorbeeld:

```
        P = 1
        .IFDF  P
A:        .
          .                  } Programmadeel A
          .
        .ENDC
        .IFDF  Q
B:        .
          .                  } Programmadeel B
          .
        .ENDC
C:        .
          .
          .
```

Betekenis: Indien P gedefinieerd is, zoals hier het geval is, wordt het programmadeel A vertaald. Is Q niet gedefinieerd, dan wordt programmadeel B niet vertaald. Programmadeel C wordt altijd vertaald.

NOTE : De conditionele programmadelen mogen genest worden hetgeen inhoudt, dat binnen een conditioneel porgrammadeel opnieuw een conditioneel programmadeel mag voorkomen.

3.8. Macro's

3.8.0. Inleiding

Er zijn twee soorten macro's te onderscheiden. Ten eerste de systeem macro's, ook wel programmed request's genoemd, behorend bij een operating systeem. Omdat voor het gebruik van deze systeem macro's gedetailleerde kennis van het operating systeem nodig is, wordt er hier niet verder op ingegaan, maar verwezen naar de digital dokumentatie, met name naar het RT11 Reference Manual. Ten tweede zijn er door de gebruiker te definieren macro's. Dit zijn compacte schrijfwijzes voor serie's instructies, die door de macrovertaler geëxpandeerd worden. Een macro aanroep en een macro definitie moeten aan bepaalde voorwaarden voldoen.

3.8.1. Macro definitie

Een macro definitie moet beginnen met een macro aanduiding. Dit gebeurt met de assembler besturingscode .MACRO. De vorm van een .MACRO statement is:

.MACRO naam, dummy argumenten lijst

waarin: **naam** de naam is van de te definiëren macro. Deze naam moet aan de eisen voldoen die aan een symbool gesteld zijn.
, een legaal scheidingsteken moet zijn.
dummy argumentenlijst, een aantal dummy symbolen is, die aangeeft hoeveel argumenten voor de macro gebruikt worden.

De macro definitie moet afgesloten worden met een .ENDM statement. Dit .ENDM statement mag gevolgd worden door de naam van de macro.

Voorbeeld van een macro: definitie.

```
.MACRO      TYPMSG,MSG              ; type een booschap
            MOV # MSG, R1
            JSR     PC, PRINTX
            .ENDM
```

3.8.2. Macro aanroep

Een macro wordt aangeroepen op dezelfde manier als de definitie heeft plaatsgevonden. Er geldt de restrictie, dat de macro in het programma gedefinieerd moet zijn, alvorens de eerste aanroep plaats vindt. De macro aanroep heeft nu de volgende gedaante:

NAAM, ARGUMENTEN

waarin: **naam** weer de naam van de macro is.
argumenten nu de werkelijke argumenten zijn waarop de macro betrekking heeft.
› scheiding tussen naam en argumenten en argumenten onderling moet plaats vinden met komma's of spaties.

Voorgeprogrammeerde MACRO's.
Programmed Requests ⇒ O.S.
Sysmac . SML.

Definitie
.MCALL mac1, mac2, ...

4.1. Gebruik standaard subroutines

Bij het gebruik van standaard subroutines worden twee fasen onderscheiden, te weten:

1e. Het programmeren van standaardsubroutines.

2e. Het samenvoegen van standaardsubroutines met het gebruikers-programma.

1e. Het programmeren van standaard subroutines.

In het overzicht van standaard subroutines op de volgende pagina's is aangegeven hoe de betreffende subroutine aangeroepen moet worden en hoe de parameters voor deze subroutine overgedragen moet worden. Aan de hand van enige voorbeeldjes zal dit gedemonstreerd worden. Verder moeten de namen van de gebruikte subroutines als globale namen gedeclareerd worden, omdat de naam niet als label in het gebruikers-programma voorkomt.
Op deze manier is het vertaalprogramma zonder verder iets over de globale namen te weten in staat het programma te vertalen.
Ter illustratie van het gebruik van subroutines uit het standaard-subroutinepakket volgen een aantal voorbeelden.

Voorbeeld 1:

```
;... VØØRBEELD ...
;EEN VIA EEN TERMINAL INGEVOERDE TEKST
;WORDT NA HET PRINTEN VAN EEN RETURN
;OF CR KARAKTER GEPRINT


RØ=%0
R1=%1
SP=%6
PC=%7

.GLOBL  PRINTX,INTEX,INTBUF

START:  RESET           ;RESET SYSTEEM
        MOV #1ØØØ,SP    ;VUL STACK-POINTER
        JSR PC,INTEX    ;VOER TEKST IN TOT CR OF RETURN
        MOV #INTBUF,R1  ;TEKST-BUFFERADRES IN R1
        JSR PC,PRINTX   ;PRINT TEKST BUFFER
        BR START        ;NAAR START VOOR HERHALING


        .END            ;EINDE PROGRAMMA
```

```
        ;VOORBEELD  ...2...
        ;TWEE INTEGER GETALLEN A EN B WORDEN VIA DE TERMINAL
        ;INGEVOERD. DE GETALLEN WORDEN NA TOT INTEGER GECON-
        ;VERTEERD TE ZIJN MET ELKAAR VERMENIGVULDIGD.
        ;HET RESULTAAT WORDT GEPRINT

        R0=%0    ;DECLARATIE REGISTERS
        R1=%1
        SP=%6
        PC=%7


.GLOBL  $ERVEC,PRINTX,PRINTB,ATOI,MUL,INTEX,INTBUF


START:  RESET            ;RESET HET SYSTEEM
        MOV #1000,SP     ;VUL STACKPOINTER
        MOV #FOUT,$ERVEC    ;VUL FOUT-VECTOR MET ADRES FOUT
        MOV #TE1,R1      ;TEKSTBUFFERADRES IN R1
        JSR PC,PRINTX    ;PRINT TEKST VOLGENS R1
        JSR PC,INTEX     ;VOER TEKST IN
        MOV #INTBUF,R1   ;SOURCE (TEKSTBUFFER) ADRES IN R1
        MOV #A,R0        ;DESTINATION ADRES IN R0
        JSR PC,ATOI      ;CONVERTEER VAN ASCII NAAR INTEGER
        BVS  FOUT        ;BIJ FOUTEN NAAR FOUT
        MOV #B,R0        ;NIEUW DESTINATION ADRES IN R0
        JSR PC,ATOI      ;CONVERTEER VOLGENDE GETAL
        BVS  FOUT        ;BIJ FOUTEN WEER NAAR FOUT
        MOV #A,R1        ;NIEUW SOURCE ADRES IS A
        MOV #B,R0        ;NIEUW DESTINATION ADRES IS B
        JSR PC,MUL       ;B:=A*B   INTEGER VERMENIGVULDIGING
        MOV #TE2,R1      ;TEKST-BUFFER ADRES IN R1
        JSR PC,PRINTX    ;PRINT TEKST
        MOV #B,R1        ;ADRES B IN R1 OM B TE PRINTEN
        JSR PC,PRINTB    ;PRINT INTEGER
        BR START         ;NAAR START

FOUT:   MOV #FTE,R1      ;TEKSTBUFFER-ADRES IN R1
        JSR PC,PRINTX    ;PRINT TEKST
        BR START         ;NAAR START,PROBEER OPNIEUW

TE1:    .BYTE 15,12      ;RETURN EN LINEFEED KARAKTERS
        .ASCII /TYPE INTEGER GETALLEN A EN B GESCHEIDEN/
        .ASCII / DOOR EEN KOMMA/
        .BYTE 15,12,0            ;RETURN,LINEFEED,EINDE TEKST
        .EVEN

TE2:    .BYTE 15,12      ;RETURN EN LINEFEED
        .ASCII /A*B = /
        .EVEN

FTE:    .BYTE 15,12      ;RETURN EN LINEFEED
        .ASCIZ /FOUT,OPNIEUW/
        .EVEN

A:      .WORD 0
B:      .WORD 0
        .END
```

```
        ;VOORBEELD  ...3...
        ;VAN EEN IN TE TYPEN FLOATING POINT GETAL WORDT
        ;DE SINUS GEPRINT

        R0=%0
        R1=%1
        SP=%6
        PC=%7

.GLOBL  INTEX,INTBUF,PRINTX,PRINTF,SINUS,CONATF

START:  RESET           ;RESET HET SYSTEEM
        MOV #1000,SP    ;VUL DE STACKPOINTER
        MOV #FOUT,$ERVEC   ;ADRES FOUT IN FOUTVECTOR
        MOV #TE1,R1     ;ADRES TEKSTBUFFER IN R1
        JSR PC,PRINTX   ;PRINT TEKST
        [illegible]
        MOV #INTBUF,R1  ;SOURCE ADRES VOOR CONVERSIE
        MOV #A,R0       ;DESTINATION ADRES BIJ CONVERSIE
        JSR PC,CONATF   ;CONVERTEER VAN ASCII NAAR FLOATING
        BVS FOUT        ;BIJ FOUTEN NAAR FOUT
        MOV #B,R0       ;DESTINATION ADRES BIJ SINUS
        MOV #A,R1       ;SOURCE ADRES BIJ SINUS
        JSR PC,SINUS    ;SINUS BEWERKING
        MOV #TXT,R1     ;TEKSTBUFFER-ADRES IN R1
        JSR PC,PRINTX   ;PRINT TEKST
        MOV #B,R1       ;ADRES RESULTAAT
        JSR PC,PRINTF   ;PRINT RESULTAAT
        BR START        ;NAAR START,OPNIEUW

FOUT:   MOV #FTE,R1     ;FOUT TEKST ADRES IN R1
        JSR PC,PRINTX   ;PRINT TEKST
        BR START        ;NAAR START,OPNIEUW

TE1:    .BYTE 15,12     ;RETURN EN LINEFEED KARAKTERS
        .ASCII /TYPE FLOATING POINT GETAL/
        .BYTE 15,12
        .ASCIZ /A: /
        .EVEN

TXT:    .BYTE 15,12     ;RETURN EN LINEFEED
        .ASCIZ /SINUS A = /
        .EVEN

FTE:    .BYTE 15,12
        .ASCIZ /FOUT,OPNIEUW/
        .EVEN

A:      .WORD  0,0
B:      .WORD 0,0

        .END
```

4.2. Het samenvoegen van standaard subroutines met het gebruikersprogramma.

De gebruiker moet ervoor zorgen, dat zijn programmadeel of -delen in vertaalde vorm, dus als z.g. objekt file aanwezig is op de schijf van het PDP 11E10 system. Op dezelfde schijf zijn ook de standaard bibliotheken LIB10, LIB20 en LIBLSI aanwezig.
Met behulp van het systeemprogramma LINK wordt nu het gebruikersprogramma samengeknoopt met een standaard bibliotheek. De procedure hiervoor is de volgende:

1e. Start het LINK programma door na het 'RT11'•teken te typen R LINK. Eventueel moet eerst RT11 gestart worden door de computer te starten op het adres 173000.
{'HALT','173000 → SR', 'LOAD ADDR', 'ENABLE', 'START'}
Het is ook mogelijk dat RT11 al werkt, doch dat een lopend programma eerst afgebroken moet worden. Dit gebeurt door 'CNTL/C' te typen.
(De 'CNTL' toets en de 'C' toets gelijktijdig te bedienen.)

2e. Nadat het LINK-programma gestart is meldt dit programma zich met een * teken. Hierna moet een commandostring van de volgende gedaante ingetypt worden:

PROG,TT: = UPROG, SUBR1, LIB20/T:1000 'CR'

Hierin is:

PROG : de naam van het programma dat na samenknopen ontstaat en dat met een R (RUN) commando geëxecuteerd kan worden. Aan dit programma wordt automatisch de extensie .SAV toegekend.

TT: : Het medium waar de z.g. linkmap wordt geprint of uitgevoerd. In de linkmap staan de adressen die aan alle gebruikte globale namen zijn toegekend.
De linkmap geeft verder informatie over de programma omvang. Wordt een linkmap niet gewenst dan kan TT: weggelaten worden. De komma moet dan ook weggelaten worden.

UPROG : Het vertaalde gebruikersprogramma, dat de extensie .OBJ moet hebben. Bij vertalen met de MACRO-vertaler op de PDP11 wordt deze extensie automatisch toegekend. Wordt gebruik gemaakt van een ponsband die verkregen is door vertalen op de DEC10, dan moet bij het plaatsen van deze ponsband op de schijf een naam met de extensie .OBJ gebruikt worden. Het op schijf plaatsen van een ponsband gebeurt met een van de systeemprogramma's PIP of LOAD.

SUBR1 : Een vertaalde subroutine, die door de gebruiker geschreven is of reeds op de schijf aanwezig was. De gebruiker kan zijn te programmeren probleem in een aantal afzonderlijk te vertalen delen opsplitsen en deze delen afzonderlijk vertalen. Met LINK worden deze delen dan samengevoegd.

LIB20 : De standaardsubroutine bibliotheek, in dit geval voor de PDP 11/20. Er ontstaat dus een programma dat op de PDP 11/20 verwerkt moet worden.

/T:1000 : Specificatie van het startadres. Deze specificatie is alleen noodzakelijk indien het hoofdprogramma als absolute sectie (d.m.v. .ASECT) is geschreven. Is dit niet het geval, dan kan de startadres specificatie weggelaten worden en wordt automatisch 1000 als startadres genomen.

'CR' : Afsluiting van de commandostring.
'CR' betekent, afhankelijk van de terminal die bij de computer aanwezig is het drukken van de 'CR' toets of de 'RETURN' toets.

Nadat de commandostring getypt is, wordt een executeerbaar programma gevormd en een linkmap geprint op de terminal. Het samenknoopproces is beeindigd als op de terminal opnieuw een '*' teken wordt geprint. Is het gemaakte programma bedoeld voor de PDP 11/20, dan moet het via de koppeling tussen de PDP 11/10 en de PDP 11/20 getransporteerd worden. De procedure hiervoor is de volgende:

Transport programma's van de PDP11 E10 naar de PDP11/20.

Type op de PDP11/20 : R FILIN 'CR'
Wacht op * teken
Type op de PDP11E10 : R FILOUT 'CR'
Wacht op * teken
Type op de PDP11/20 : PROG.SAV = 'CR'
Type op de PDP11E10 : PROG.SAV 'CR'
Wacht tot beide computers een * teken geprint hebben.

NB: Alvorens R FILIN of R FILOUT getypt kan worden moet misschien een nog lopend programma met 'CNTL/C' onderbroken worden.
Nadat op beide computers een '*' teken geprint is, kunnen door het intypen van andere filenamen nog meerdere programmatransporten plaatsvinden.

4.3. Gebruik FOUTVECTOR.

Bij een aantal routines wordt, indien er iets fout gaat, gesprongen naar de locatie die in de foutvector is geplaatst.
De foutvector heeft de globale naam: $ERVEC.
Bij alle routines die bij fouten naar de locatie in de foutvector springen is hiervan melding gemaakt. Het gebruik van de foutvector wordt met onderstaand voorbeeld verduidelijkt. Bij gebruik van de LSI11 is een extra actie nodig. Na het vullen van de foutvector moet hier de subroutine ERRSET aangeroepen worden.

Voorbeeld gebruik foutvector.
In een programma komt een floating point deling voor, waarbij eventueel delen door 0 kan voorkomen. Is dit het geval, dan moet

het programma onderbroken worden en moet een foutmelding geprint worden.

Programma:

```
        .GLOBL    DIVFLT,$ERVEC
          .
          .
        MOV   #FOUT,SERVEC          ; ADRES FOUT IN FOUT-VECTOR
          .
          .
        JSR   PC,DIVFLT             ; DE DEELOPDRACHT
          .
          .
          .
        HALT                        ; EINDE NORMALE PROGRAMMA

FOUT:   MOV   #FOUTXT,R1            ; FOUTPROGRAMMA
        JSR    PC,PRINTX            ; PRINT FOUTBOODSCHAP
        JMP    VERVOL               ; VERVOLG PROGRAMMA

FOUTXT: .BYTE 15.12
        .ASCIZ/DELING DOOR 0/
```

4.4. Overzicht en beschrijving van aanwezige subroutines.

De in de standaardbibliotheken aanwezige subroutines worden in de nu volgende lijst gegeven. Naast een korte functie omschrijving wordt aangegeven hoe de subroutine aangeroepen moet worden en hoe de variabelen aan de subroutine moeten worden doorgegeven. De lijst is onderverdeeld in een aantal groepen, zoals tekstroutines, conversie-routines, floating point routines etc.

4.4.1. Tekst routines

Subroutine naam:	PRINTX
Subroutine aanroep:	JSR PC,PRINTX
Variabelen transport:	Adres tekstbuffer in R1.
Functieomschrijving:	De subroutine PRINTX print een buffer met ASCII-karakters op de terminal. Het printen stopt wanneer een 0-byte wordt ontmoet. Elke tekstbuffer moet dus met een byte gelijk aan 0 afgesloten worden.

Subroutine naam:	INTEX
Subroutine aanroep :	JSR PC,INTEX
Variabelen transport:	Tekst wordt geplaatst in buffer INTBUF.
Functie omschrijving:	De subroutine INTEX leest karakters in van het terminal keyboard en plaatst deze in de buffer genaamd INTBUF. Het inlezen van karakters stopt zodra een 'CR' karakter wordt getypt. Maximaal kunnen 80 karakters ingetypt worden. Alle ingevoerde karakters worden tevens geprint. Er bestaan twee speciale karakters, n.l. - RUBOUT, voor het verwijderen van telkens het laatste karakter. - CNTL/U, voor het verwijderen van alles wat reeds ingevoerd is. (Complete regel) (CNTL/U betekent het gelijktijdig bedienen van de 'CNTL'toets en de 'U'toets.

Subroutine naam:	PRINTI
Subroutine aanroep:	JSR PC,PRINTI
Overdracht variabelen:	Adres te printen integer in R1 Aantal karakters waarmee integer geprint wordt in R0.
Functie omschrijving:	De subroutine PRINTI print een integer waarde op de terminal.

Subroutine naam:	PRINTB
Subroutine aanroep:	JSR PC,PRINTB
Overdracht variabelen:	Adres te printen integer in R1.
Functie omschrijving :	De subroutine PRINTB print de integer waarvan het adres in R1 staat op de terminal. Het aantal karakters waarmee de integer geprint wordt is vast en bedraagt 6.
Subroutine naam:	PRINTF
Subroutine aanroep:	JSR PC,PRINTF
Overdracht variabelen:	Adres te printen floating point getal in R1
Functie omschrijving :	De subroutine PRINTF print het floating point getal waarvan het adres in R1 staat op de terminal. Het formaat waarin het floating point getal geprint wordt is: - totaal aantal karakters: 14 - aantal karakters Mantisse: 5 - aantal karakters Exponent: 2
Subroutine naam:	PRKAR
Subroutine aanroep:	JSR PC,PRKAR
Overdracht variabelen:	Het te printen karakter (ASCII-code) moet in R0 geplaatst zijn.
Functie omschrijving:	De subroutine PRKAR print het karakter in R0 op de terminal.
Subroutine naam:	ATOI
Subroutine aanroep:	JSR PC,ATOI
Overdracht variabelen:	1e Adres te converteren ASCII-string in R1. Adres integer resultaat in R0.
Functieomschrijving:	De subroutine ATOI converteert een ASCII-string naar een integer getal, waarbij komma's en 'CR' of 'RETURN' karakters als scheidingstekens fungeren. Is een string niet tot een integer getal te converteren, b.v. als een letter voorkomt, dan is het resultaat 0 en wordt het V-bit in het processor status-woord gezet.

Na de conversie wijst R1 naar het karakter volgend op het scheidingsteken, zodat twee getallen gescheiden door een komma na elkaar geconverteerd kunnen worden zonder R1 aan te passen.

Subroutinenaam:	ITOA
Subroutine aanroep:	JSR PC,ITOA
Overdracht variabelen:	Adres integer getal in R1. Adres ASCII resultaat in R0.
Functieomschrijving:	De subroutine ITOA converteert het integer getal, waar-van het adres in R1 staat, naar een ASCII-string bestaande uit 6-karakters en plaatst deze string in het geheugen te beginnen bij de locatie die in R0 aangegeven is.

4.4.2. Conversie Routines.

Subroutinenaam:	FIX
Subroutineaanroep:	JSR PC,FIX
Overdrachtsvariabelen:	Adres te converteren floating point getal in R1. Adres integer resultaat in R0.
Functieomschrijving: (R0):=FIX(R1)	De subroutine FIX converteert het floating point getal waarvan het adres in R1 staat naar een integer getal waarvan het adres in R0 staat. De omzetting vindt plaats door middel van afronding.
Subroutinenaam:	FIXB
Subroutineaanroep:	JSR PC,FIXB
Overdracht variabelen:	Adres te converteren floating point getal in R1. Adres integer resultaat in R0.
Functie omschrijving: (R0):=FIXB (R1)	De subroutine FIXB converteert het floating point getal waarvan het adres in R1 staat naar een integer getal waarvan het adres in R0 staat. De omzetting vindt plaats door middel van afbreking.
Subroutinenaam:	FLOT
Subroutineaanroep:	JSR PC,FLOT
Overdracht variabelen:	Adres te converteren integer in R1. Adres floating point resultaat in R0.
Functieomschrijving: (R0): =Flot (R1)	De subroutine FLOT converteert het integer getal, waarvan het adres in R1 staat, naar een floating point getal waarvan het adres in R0 staat.
Subroutinenaam:	CONATF
Subroutineaanroep:	JSR PC, CONATF
Overdracht variabelen:	le adres te converteren ASCII-string in R1. Adres floating point resultaat in R0.
Functieomschrijving:	De subroutine CONATF converteert een ASCII string, waarvan het le adres in R1 staat

naar een floating point getal waarvan het adres in R0 staat. Komma's en 'CR' of 'RETURN' karakters dienen als scheidingstekens.
Is conversie niet mogelijk, dan wordt het V-bit in het processor status-woord gezet.
Na de conversie wijst R1 naar het karakter volgend op het scheidingsteken, zodat na elkaar twee getallen gescheiden door een komma, geconverteerd kunnen worden zonder dat R1 aangepast hoeft te worden.

Subroutinenaam:	CONFTA
Subroutine aanroep:	JSR PC,CONFTA
Overdracht variabelen:	Adres te converteren floating point getal in R1. Adres ASCII-resultaat in R0.
Functie omschrijving:	De subroutine CONFTA converteert het floating point getal, waarvan het adres in R1 staat naar een ASCII-string waarvan het eerste adres in R0 staat. Het resultaat bestaat uit 14 ASCII karakter. Afhankelijk van de grootte van het te converteren getal wordt de E-notatie gebruikt.

4.4.3. Integer routines.

Subroutine naam:	MUL
Subroutine aanroep:	JSR PC,MUL
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functie omschrijving :	De subroutine MUL vermenigvuldigt de integer aangewezen door R1 met de integer aangewezen door R0 en plaatst het resultaat op de plaats aangewezen door R0. Is het resultaat te groot voor 1 woord locatie, dan wordt via de foutvector $ERVEC het programma voortgezet.
Subroutine naam:	DIV
Subroutine aanroep:	JSR PC,DIV
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functie omschrijving: (R0):= (R0)/(R1)	De subroutine DIV deelt de integer aangewezen door R0 door de integer aangewezen door R1 en plaatst het resultaat in de locatie aangewezen door R0. Bij overflow wordt gesprongen via de foutvector $ERVEC.

4.4.4. Floating Point Routines

Subroutine naam:	ADDFLT
Subroutine aanroep:	JSR PC,ADDFLT
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functiebeschrijving: (R0):=(R1)+(R0)	De subroutine ADDFLT telt de floating point getallen aangewezen door resp. R0 en R1 op en plaatst het resultaat op de locatie aangewezen door R0. Bij overflow wordt gesprongen via de foutvector $ERVEC.
Subroutine naam:	SUBFLT
Subroutine aanroep:	JSR PC,SUBFLT
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functie beschrijving: (R0):=(R0)-(R1)	De subroutine SUBFLT trekt het floating point getal aangewezen door R1 af van het floating point getal aangewezen door R0 en plaatst het resultaat in de locatie aangewezen door R0. Bij overflow wordt gesprongen via de foutvector $ERVEC.
Subroutinenaam:	NEGFLT
Subroutineaanroep:	JSR PC,NEGFLT
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functie beschrijving (R0)=-(R1)	De subroutine NEGFLT vult de locaties aangewezen door R0 met minus het floating point getal aangewezen door R1.
Subroutine naam:	CMPFLT
Subroutineaanroep:	JSR PC,CMPFLT
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functie beschrijving: (R0)-(R1)	De subroutine CMPFLT vergelijkt source en destination en vult het processor status woord

afhankelijk van het resultaat. Hierna kunnen direkt conditionele sprongen, b.v. BGT, opgenomen worden.

Subroutinenaam:	MULFLT
Subroutineaanroep:	JSR PC,MULFLT
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functie beschrijving: (R0):=(R0)*(R1)	De subroutine MULFLT vermenigvuldigt de floating point getallen aangewezen door R0 en R1 en plaatst het resultaat op de locaties, aangewezen door R0. Bij overflow sprong via de foutvector $ERVEC.
Subroutinenaam:	DIVFLT
Subroutineaanroep:	JSR PC,DIVFLT
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functie beschrijving: (R0):=(R0)/(R1)	De subroutine DIVFLT deelt het floating point getal aangewezen door R0 door het floating point getal aangewezen door R1. Het resultaat wordt op de locatie aangegeven door R0 geplaatst. Bij overflow wordt gesprongen via de foutvector $ERVEC.
Subroutinenaam:	SINUS
Subroutineaanroep:	JSR PC,SINUS
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functiebeschrijving: (R0):= SIN ((R1))	De subroutine SINUS plaatst de sinus van het floating point getal aangewezen door R1 op de locaties aangegeven door R0. Bij overflow tijdens de berekeningen wordt gesprongen via de foutvector $ERVEC.
Subroutinenaam:	COSIN
Subroutineaanroep:	JSR PC,COSIN
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functiebeschrijving: (R0):= COSIN ((R1))	De subroutine COSIN plaatst de cosinus van het floating point getal aangewezen door R1 op de locaties aangewezen door R0. Bij Overflow tijdens de berekeningen wordt

gesprongen via de foutvector $ERVEC.

Subroutinenaam:	ARCTNG
Subroutineaanroep:	JSR PC,ARCTNG
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functiebeschrijving: (R0):=ARCTNG(R1)	De subroutine ARCTNG plaatst de arcus tangens van het floating point getal aangewezen door R1 op de locaties aangewezen door R0. Bij overflow tijdens de berekeningen wordt gesprongen via de foutvector $ERVEC.
Subroutinenaam:	EXPFLT
Subroutineaanroep	JSR PC,EXPFLT
Overdracht variabelen:	Source adres in R1. Destination adres in R0.
Functiebeschrijving: (R0) := EXP ((R1))	De subroutine EXPFLT bepaalt de e-macht van het floating point getal aangewezen door R1 en plaatst dit op de locaties aangewezen door R0. Bij overflow tijdens de berekeningen via de foutvector $ERVEC.
Subroutinenaam:	LOG
Subroutineaanroep	JSR PC,LOG
Overdracht variabelen:	Source adres in R0. Destination adres in R1.
Functiebeschrijving: (R0):=LN(R1)	De subroutine LOG bepaalt de natuurlijke logaritme van het floating point getal aangewezen door R1 en plaatst het resultaat op de locaties aangewezen door R0. Bij overflow tijdens de berekeningen wordt gesprongen via de foutvector $ERVEC.
Subroutinenaam:	LOG1Ø
Subroutineaanroep:	JSR PC,LOG1Ø
Overdracht variabelen:	Source adres in R0. Destination adres in R1.
Functiebeschrijving: (R0):= 10LOG((R1))	De subroutine LOG1Ø bepaalt de 10^{o} logaritme van het floating point getal aangewezen door R1 en plaatst het resultaat in de locaties aangewezen door R1 en plaatst het resultaat in de locaties aangewezen door R0. Bij overflow wordt gesprongen via de foutvector $ERVEC.

Subroutinenaam:	SQROOT
Subroutineaanroep:	JSR PC,SQROOT
Overdracht variabelen:	Source adres in R0.
	Destination adres in R1.
Functiebeschrijving:	De subroutine SQROOT bepaalt de wortel van het floating point getal aanwezen door R1 en plaatst het resultaat op de locaties aangewezen door R0. Bij fouten wordt gesprongen via de foutvector $ERVEC.

4.4.5. Hulproutines.

Subroutinenaam:	SAVREG
Subroutineaanroep:	JSR PC,SAVREG
Overdracht variabelen:	Geen
Functiebeschrijving :	De subroutine SAVREG redt de registers R0 t/m R5 op de STACK. De stackpointer R6 moet dus met een legale waarde gevuld zijn. De registers R0 t/m R5 kunnen met behulp van de subroutine BACKRG weer teruggehaald worden.
Subroutinenaam:	BACKRG
Subroutineaanroep:	JSR PC,BACKRG
Overdrachtvariabelen:	Geen
Functiebeschrijving:	De subroutine BACKRG restored de registers R0 t/m R5 die m.b.v. de subroutine SAVREG op de STACK geplaatst zijn.
Subroutinenaam:	ERRSET
Subroutineaanroep:	JSR PC,ERRSET
Overdracht variabelen:	Geen
Functiebeschrijving:	De subroutine ERRSET zorgt voor de opvang van fouten die optreden tijdens FIS-instrukties (FIS=Floating point Instructie Set). Alleen de LSI 11 maakt gebruik van FIS-instrukties.

Subroutinenaam:	WACHT
Subroutinenaam:	JSR PC,WACHT
Overdracht variabelen:	R0 bevat aantal wachteenheden.
Functiebeschrijving:	De subroutine WACHT is een "inline" software wachtroutine, die een aantal wachteenheden volgens R0 wacht. Een wacht-eenheid duurt ongeveer ..

4.4.6. Display routines. ook door Fortran te gebruiken.

Subroutinenaam:	DISPNT
Subroutineaanroep:	JSR PC,DISPNT
Overdrachtvariabelen:	R0 wijst naar X-coördinaat. R1 wijst naar Y-coördinaat.
Functiebeschrijving:	De subroutine DISPNT vertoont het punt op de displayscope, waarvan R0 en R1 de coördinaten aanwijzen. Voor de coördinaten geldt: $-2.040<x,y<2.048$.
Subroutinenaam:	DISLIN
Subroutineaanroep:	JSR PC,DISLIN
Overdracht variabelen:	R1 wijst naar het veld met de coördinaten (x_0,y_0,x_1,y_1).
Functiebeschrijving:	De subroutine DISLIN vertoont op de display-scope een lijn, die ligt tussen de punten (x_0,y_0) en $x_1,y_1)$. Als begrenzing geldt: $-2048 < x_0,y_0,x_1,y_1<2048$
Subroutinenaam:	CLRSCP
Subroutineaanroep:	JSR PC,CLRSCP
Overdrachtvariabelen:	Geen
Functiebeschrijving:	De subroutine CLRSCP veegt het scherm van de Display-scope schoon.
Subroutinenaam:	DISKAR
Subroutineaanroep:	JSR PC,DISKAR
Overdrachtvariabelen:	Karakter in R0
Functieomschrijving:	De subroutine DISKAR vertoont op de display scope het karakter, waarvan de ASCII code in R0 is geplaatst. Het karakter wordt als een 5x7 punt matrix weergegeven. Komt de display buiten het scherm, dan wordt het V-bit geset.
Subroutinenaam:	DISTXT
Subroutineaanroep:	JSR PC,DISTXT
Overdrachtvariabelen:	Adres tekstbuffer in R1
Functiebeschrijving:	De subroutine DISTXT vertoont op de displayscope de tekstbuffer waarvan het adres in R1 is geplaatst. De tekstbuffer moet eindigen met een 0 byte (bytepositie die geheel 0 is). Wanneer het display scherm vol raakt wordt het V-bit geset.

4.4.7. Plotroutines

ook door Fortran te gebruiken.

Subroutinenaam:	PLTPNT
Subroutineaanroep:	JSR PC,PLTPNT
Overdracht variabelen:	R0 wijst naar X-coördinaat R1 wijst naar Y-coördinaat
Functiebeschrijving:	De subroutine PLTPNT plot op de XY-recorder het punt waarvan de coördinaten aangewezen worden door R0 en R1. Voor de coördinaten geldt: $-2048 \leqslant X,Y < 2048$

Subroutinenaam:	PLTLIN
Subroutineaanroep:	JSR PC,PLTLIN
Overdracht variabelen:	R1 wijst naar het veld met de coördinaten X0,Y0,X1,Y1.
Functiebeschrijving:	De subroutine PLTLIN plot op de XY-recorder een lijn tussen de punten (X0,Y0) en (X1,Y1). Als begrenzing geldt: $-2048 \leqslant X0,Y0,X1,Y1 < 2048$.

Subroutinenaam:	PLTKAR
Subroutineaanroep:	JSR PC,PLTKAR
Overdracht variabelen:	Te plotten karakter in R0.
Funtiebeschrijving:	De subroutine PLTKAR plot het karakter waarvan de ASCII code in R0 geplaatst is op de XY-recorder. Het karakter wordt weergegeven in een 5x7 puntmatrix. Wanneer de plot buiten de toegestane afmetingen raakt wordt het V-bit geset.

Subroutinenaam:	PLTTXT
Subroutineaanroep:	JSR PC,PLTTXT
Overdracht variabelen:	Adres tekstbuffer in R1.
Functiebeschrijving:	De subroutine PLTTXT plot de tekstbuffer, waarvan het adres in R1 staat, op de XY-recorder. De tekstbuffer moet eindigen met een 0-byte. (Byte positie die geheel 0 is.) Wanneer een plot vol-raakt wordt het V-bit geset.

HOOFDSTUK 5: INTRODUCTIE VAN HET RT11 OPERATING SYSTEEM

5.0. Inleiding

Een operating systeem kan beschouwd worden als een software interface tussen een gebruiker resp. gebruikersprogramma en de hardware van een computer systeem. De faciliteiten die door een operating systeem geboden worden zijn:

1° Opslag en verwerking van programma's.
2° Hulpprogramma's
3° Toegang tot randapparatuur.

Alhoewel het RT11 systeem de mogelijkheid heeft een dubbele job te verwerken, wordt enkel de single job versie besproken. In de eenvoudige toepassingen zal zeker met deze single job versie volstaan kunnen worden.
RT11 is een klein real time operating systeem. Het is klein omdat het permanent in het geheugen aanwezige deel slechts 1600 computerwoorden bedraagt. Het is real time omdat snelle reaktie op informatie van randapparaten mogelijk is.

5.1.0. RT11 monitor

De funkties van de monitor zijn:

– *Verzorging van de communicatie tussen operator en systeem.*
– *Het laden van programma's in het geheugen en het verwerken.*
– *Verzorging van in/uitvoer en speciale funkties.*

De monitor bestaat uit een resident gedeelte, dat permanent in het kerngeheugen aanwezig is (RMON) en een deel dat indien nodig, door "swappen" (uitwisselen) in het geheugen geplaatst wordt (KMON). Bij swappen wordt een gedeelte van het kerngeheugen op massa geheugen geplaatst om plaats te maken voor het gewenste monitordeel. Wanneer dat monitordeel niet meer nodig is, wordt de oorspronkelijke geheugeninhoud terug geplaatst.
In fig. 17 zijn de monitorfuncties in beeld gebracht. Men kan op twee manieren toegang krijgen tot de monitor: 1° vanaf de terminal;
2° vanuit een programma d.m.v. zogenaamde programmed requests.

De monitor kan op twee manieren aangeven dat gegevens vanaf de terminal verwacht worden.

1° Met het . (punt) karakter geeft de monitor aan, dat aan de monitor een commando, b.v. een RUN commando, kan worden gegeven.
2° Met het * (asterisk) karakter geeft een systeem programma aan, dat een file specificatie wordt verwacht.

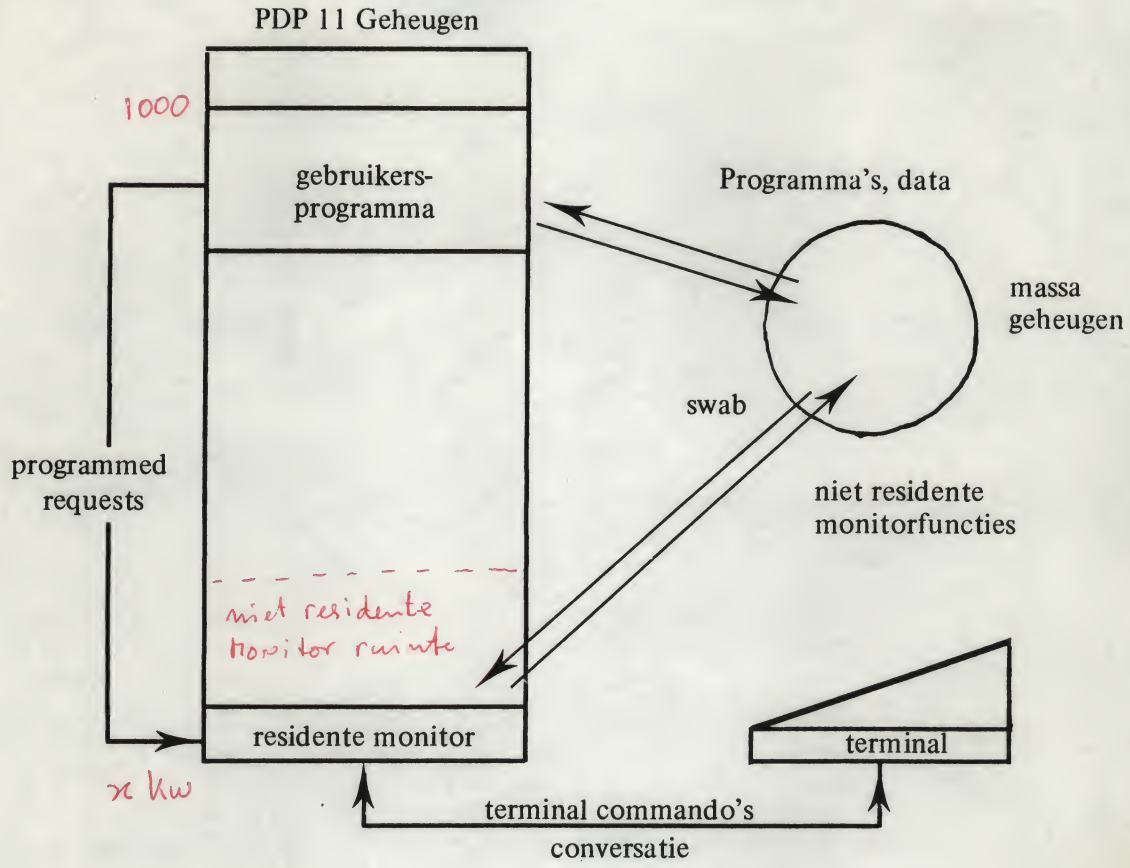


fig. 17 Schets van de monitorfuncties.

5.1.1. Algemene file specificatie

Alvorens over te gaan tot de beschrijving van de file specificatie, zal het begrip file nader bekeken worden.
Een file (bestand) is een verzameling computerwoorden die als een eenheid behandeld wordt bij invoer, opslag en uitvoer. De behandeling van de file als eenheid wordt mogelijk door aan de file een unieke naam toe te kennen. Om snel toegang te krijgen tot een file is op het massageheugen een *directory (catalogus)* opgebouwd. Hierin wordt door het RT11 operating systeem bijgehouden, welke files op het betreffende medium aanwezig zijn en waar precies (fysisch) de gegevens gevonden kunnen worden.
Om te voorkomen, dat elke keer dat toegang (access) tot een file nodig is, de gehele directory in het werkgeheugen gelezen moet worden om de plaats van de file op te zoeken, wordt gebruik gemaakt van het openen en sluiten van files. Een geopende file kan in een draaiend programma, zonder dat de directory gelezen behoeft te worden, gebruikt worden om data in te schrijven of uit te lezen.

Filespecificatie

In het algemeen moeten aan systeemprogramma's filespecificaties in onderstaande vorm worden gegeven. Er is daarbij sprake van een input- en een output-file.

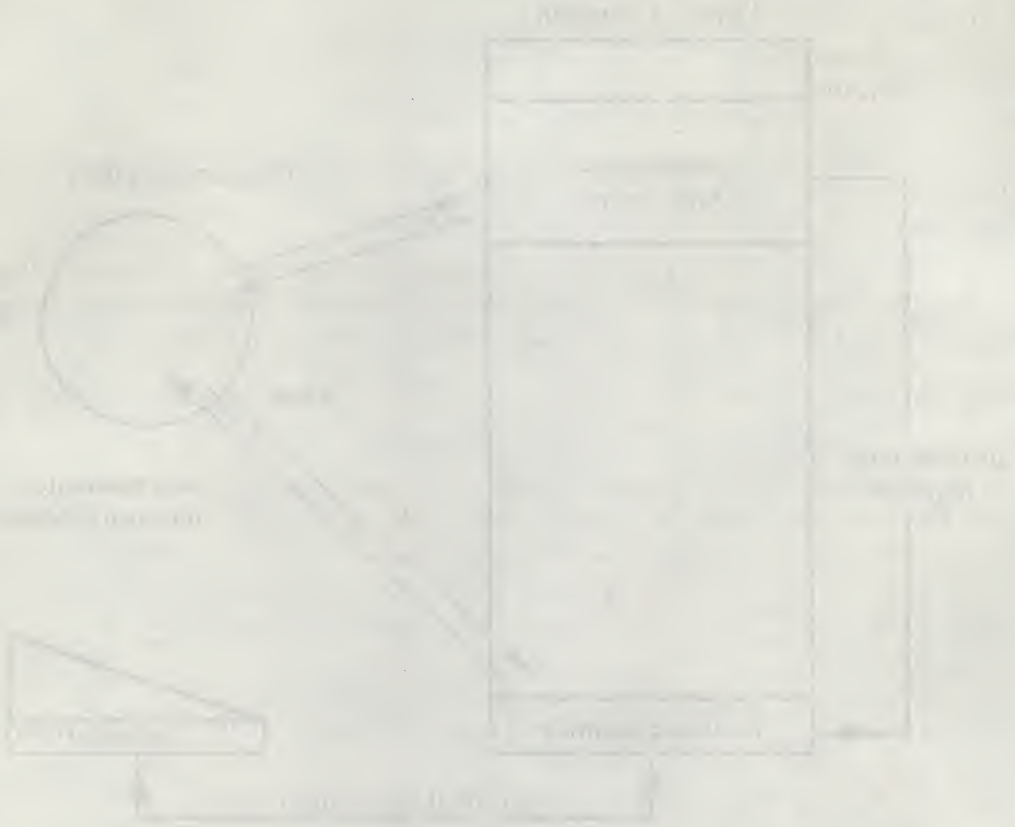
OUTPUT = INPUT

uitgewerkt:

DEV: FILNAM. EXT = DEV: .FILNAM .EXT

Voorbeeld: DT1: NAAM. MAC = DTO: MAAN. MAC

waarin: DEV: de standaard afkorting van een randapparaat.



Hieronder zijn enige voorbeelden gegeven:

RKO:	schijfunit 0
TT:	terminal
PR:	high speed ponsband lezer
PP:	high speed ponsband ponser
DTn:	Dec tape unit n
CTn:	Cassette tape unit n

Wanneer het systeem device gebruikt wordt, mag de apparaataanduiding weggelaten worden. Het systeem device is het massamedium waarop het operating systeem aanwezig is (PDP11 E 10 – schijfunit 0 , PDP11/20 – magneetbandunit 0).

FILNAM is een naam van maximaal zes karakters die aan een file wordt (is) toegekend.

Extensies: .EXT

Extensies zijn uitbreidingen van de filenaam, die veelal het type file aangeven. Een aantal systeemprogramma's levert files met standaard extensies. Een aantal van deze standaardextensies zijn in onderstaande tabel gegeven:

.MAC	Macro of assembler source programma
.FOR	Fortran source programma
.BAS	Basic source programma
.DAT	Basic of Fortran datafile
.LDA	Absolute load file (kan op een kale PDP11 gedraaid worden)
.LST	Listing file
.MAP	Link map file
.OBJ	Vertaalde, nog te linken file (output van macro resp. fortran vertaler)
.SAV	Executeerbaar programma geschikt voor RUN commando (output van link programma)
.SYS	systeemprogramma's.

5.1.2. Starten RT11

Voor het opstarten van het RT11 operating systeem is op de computer (beide) een z.g. bootstrap loader aanwezig. Dit is een stukje programma, uitgevoerd in Read only geheugen, waarmee de RT11 monitor van het massageheugen wordt gehaald en gestart wordt. De bootstrap loader wordt op de volgende manier gestart:

1. De computer moet aan staan.
2. Het massageheugen moet op de juiste manier aanstaan.
3. Plaats het start-adres van de bootstrap loader in het schakelaar-register. Dit adres is 173000.
4. Zet de computer op halt, druk load addr.
5. Zet de computer op enable, druk start.

De RT11 monitor wordt nu geladen en na een tijdje meldt het RT11 systeem zich via de terminal. Na het . teken kan nu een commando aan de monitor gegeven worden, b.v. het invoeren van de datum: .DAT 1-JAN-75

5.1.3. Monitor commando's

Normaal gesproken worden RT11 monitor commando's gegeven na een .karakter. Een uitzondering hierop wordt gevormd door speciale keyboard functies, die op interruptie-basis op elk moment kunnen worden gegeven. Deze keyboard functies zijn:

CNTL/S Halt Print dus niet onderdrukken.
CNTL/Q Continue Print

CNTL/C (wordt getypt door het ingedrukt houden van de CNTL toets en daarbij de letter c te typen).
Op de terminal wordt dit karakter geprint als ↑ C. Met dit karakter kan een lopend programma onderbroken worden en vindt terugkeer naar de monitor plaats. Is het te onderbreken programma bezig met een I/0 transfer, dan moet twee keer een CNTL/C worden gegeven, alvorens afbreking plaats vindt.

CNTL/O Onderdrukt het printen op de terminal. Na een tweede CNTL/O wordt het printen voortgezet. Het CNTL/O karakter wordt geprint als ↑ O.

RUBOUT Verwijdert het laatst getypte karakter. Het verwijderde karakter (eventueel meerdere na elkaar) wordt geprint.

CNTL/U Verwijdert de totale ingevoerde tekstregel. delete

Er volgt nu een beschrijving van de meest gebruikte monitor commando's. Voor een complete lijst wordt naar de RT11 literatuur verwezen (b.v. RT11 Reference Manual).

DATE (DAT) Met dit commando wordt de actuele datum in het systeem ingevoerd. Deze datum wordt o.a. gebruikt bij het creëeren van nieuwe files, zodat later terug te vinden is wanneer een file voor het laatst gemaakt (gewijzigd) is.
Voorbeeld: . DAT 2 – JAN – 75 (return)
(voor de maand moeten de eerste drie letters van de engelse maandnaam gebruikt worden).

INITIALIZE (IN) Hiermee wordt het systeem geïnitialiseerd.
Voorbeeld: . IN (return)

ASSIGN (ASS) Wordt gebruikt om in programma's gespecificeerde apparaatnamen aan (andere) randapparaten toe te kennen.
Voorbeeld: . ASS DT 1 DK (return)
Als het programma een verwijzing naar DK bevat, dan wordt in plaats daarvan DT 1 gebruikt.
. ASS (return)
heft alle voorgaande ASSIGN's op.

CLOSE (CLO) Sluit alle geopende files.
Voorbeeld: .CLO (return)

GET (GET) Laadt een (SAV) file in het kerngeheugen. Het programma wordt nog niet gestart.
Voorbeeld: . GET DT 1 : PIET (return)

START (STA) Start een programma vanaf de bij het commando opgegeven locatie.
Voorbeeld: .STA 1000 (return)

RUN (R) Laadt een file in het kerngeheugen en start vanaf een startadres.
Voorbeeld: .R PROG (return)
Wanneer een programma zich niet op de systeem unit bevindt (b.v. DT 1), moet het runcommando volledig getypt worden.
Voorbeeld: . RUN DT 1: VOORB (return)

RE–ENTER (RE) Start een programma dat zich in het kerngeheugen bevindt, vanaf zijn restart locatie. Het RE–ENTER adres moet geplaatst zijn in locatie: 40. Het RE–ENTER bit in het Job Status woord moet 1 zijn. (Job Status Word = 44).
Voorbeeld: . RE (return)

SAVE (SAV) Plaats een gespecificeerd geheugen gebied in een file in SAV formaat. Deze file kan dan later b.v. met een RUN commando gebruikt worden.
Voorbeeld: . SAV PROGR 0–10000 (return)
Het geheugengebied vanaf adres 0 tot adres 10000 (octaal) wordt op de systeemunit geplaatst onder de filenaam: PROGR.SAV.

DEPOSIT (D) Plaatst data in het geheugen vanaf gespecificeerde locatie.
Voorbeeld: D 1000 = 1, 2, 3, 4
Op de locaties 1000 en verder worden de waardes 1 t/m 4 geplaatst.

EXAMINE (E) Print de inhoud van gespecificeerde geheugenlocatie (s).
Voorbeeld:

E 1000	(print inhoud locatie 1000)
E 1000 – 1010	(print inhoud locaties 1000 t/m 1010)

5.2. Systeemprogramma's

Voor programma-ontwikkeling zijn systeemprogramma's onder RT11 beschikbaar. We onderscheiden:

1° Assembler vertaalprogramma's : MACRO, ASSEMBLER, EXPAND
2° Hogere taal vertalers : FORTRAN, BASIC
3° Samenknoop programma's : LINK, LIBR
4° Hulpprogramma : ODT
5° Tekst manipulatie programma : EDIT
6° File manipulatie programma : PIP peripheral interface Program.

De systeemprogramma's worden hierna in de hiervoor gegeven volgorde beschreven. Er worden slechts de belangrijkste eigenschappen genoemd. Voor gedetailleerde informatie wordt verwezen naar de RT11 system reference manual.

5.2.1. Assembler vertaalprogramma's: MACRO, ASSEMBLER, EXPAND.

Assembler vertaalprogramma's worden gebruikt om in assemblertaal geschreven source programma's te vertalen, maar z.g. object modules. Object modules zijn machinecodeprogramma's die nog niet aan een absolute plaats zijn gebonden. Dit gebeurt met de linker.
Meestal kan zonder meer gebruik gemaakt worden van het vertaalprogramma MACRO. Alleen op PDP11 computers met minder dan 16 k geheugen moet gebruik gemaakt worden van de combinatie EXPAND/ASSEMBLER.
Met het programma EXPAND worden binnen het sourceprogramma de macro's geëxpandeerd. Met ASEMBL vindt hierna de vertaling plaats.
Er wordt er nu verder van uitgegaan, dat gebruik gemaakt kan worden van het MACRO programma. Dit programma kan onder RT11 geladen en gestart worden met een RUN commando van de volgende gedaante:

. R MACRO (return)

Nadat het MACRO programma gestart is, wordt vanuit dit programma om file specificatie gevraagd.

Na het * teken moet de volgende string getypt worden:

```
* OUTFIL, LIST = INFIL / S  (return)
```

Hierin is:

OUTFIL	de file specificatie voor de output (object module) van de vertaling, b.v.: DT 1 : PROG. De extensie .OBJ wordt automatisch toegevoegd.
,LIST	de file specificatie of het device waar de listing van de vertaling naar toe moet, b.v.: ,TT: voor een listing op de teletype. Wanneer niets opgegeven wordt, ontstaat geen listing.
=	Scheidt in- en output specificaties.
INFIL	de file specificatie van de bij de vertaling betrokken file(s). Dit mogen meerdere files zijn, gescheiden door komma's. Voorbeeld: PROG1 .MAC, PROG2 .MAC. De extensie .MAC mag eventueel weggelaten worden.
/S	Speciale functie switch. Hiermee kan aangegeven worden, wat wel en wat niet gelist moet worden.

Voorbeeld:

```
.R MACRO (return)
* TEST, TT:=TST1, TST2, TST3 (return)
```

5.2.2. Hogere taal vertalers: FORTRAN en BASIC

1° Het FORTRAN vertaalprogramma kan geladen en gestart worden met een RUN commando op de volgende manier:

```
. R FORTRAN
```

Hierna wordt door het FORTRAN vertaalprogramma om in en output filespecificaties gevraagd. De file specificaties moeten in de volgende gedaante gegeven worden:

```
* OUTPUT, LIST = INPUT 1 .FOR, INPUT 2 .FOR / S
```

Voorbeeld:

```
.R FORTRAN (return)
* TEST, TT: = TST1, TST2, TST3 (return)
```

In appendix C wordt een overzicht gegeven van FORTRAN op de PDP11.

2° Het BASIC programma is een interpretatief vertaalprogramma, hetgeen inhoudt, dat het programma op het moment van executie regel voor regel vertaald wordt. Onder het BASIC programma kan een programma ingetypt en geëxecuteerd worden. Het is mogelijk BASIC programma's op het massageheugen te plaatsen en op een later tijdstip onder BASIC te laden. Het BASIC vertaalprogramma wordt geladen en gestart met het commando:

```
.R BASIC
```

In appendix D wordt een overzicht gegeven van BASIC op de PDP11.

5.2.3. Samenknoopprogramma's: LINK en LIBR

1° HET LINK PROGRAMMA

Het link programma knoopt object modules, outputs van vertaal-programma's samen tot machinecodeprogramma's, die klaar zijn voor executie. Het LINK programma vervult daarbij de volgende functiess:

- Herplaatsen van elk object module en toewijzing van absolute adressen.
- Verbindt modules met gelijke globale symbolen, die in het ene module gedefinieerd zijn en in andere modules gebruikt worden.

- Maakt een besturingsblok (startadres, job status word etc.) voor het gelinkte programma.
- Kan in aangegeven bibliotheken naar nog niet gevonden globale symbolen zoeken.
- Kan een overlay structuur verzorgen.
- Maakt een loadmap (overzicht van de geheugenindeling van het gelinkte programma).

Het programma wordt aangeroepen met: .R LINK
Hierna moeten in- en output files gespecificeerd worden in de volgende gedaante:

* OUTPUT, MAP = INPUT1, INPUT2, INPUT3 / S

Hierin is:

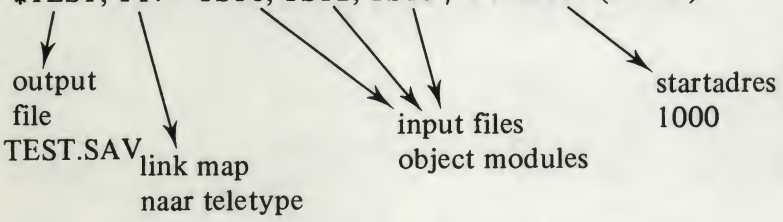
OUTPUT	file specificatie voor het output machinecodeprogramma
,MAP	file specificatie of device waar de link-map naartoe moet.
INPUT1, INPUT2, INPUT3	file specificaties van de te linken object modules (.OBJ mag weggelaten worden)
/ S	functie schakelaar.

Een aantal voor gebruik in aanmerking komende schakelaarfuncties zijn:

/ A	zet de loadmap entries op alfabetische volgorde
/ C	de commandostring wordt op de volgende regel voortgezet
/ F	specificatie van de FORTRAN bibliotheek
/ L	machinecode output in absolute load formaat
/ O : n	overlay structuur specificatie
/ T : n	specificatie van het startadres n.

Voorbeeld van een commando string om een programma te linken, waarbij startadres 1000 gespecificeerd wordt:

```
.R LINK (return)
*TEST, TT: = TST1, TST2, TST3 / T : 1000  (return)
```

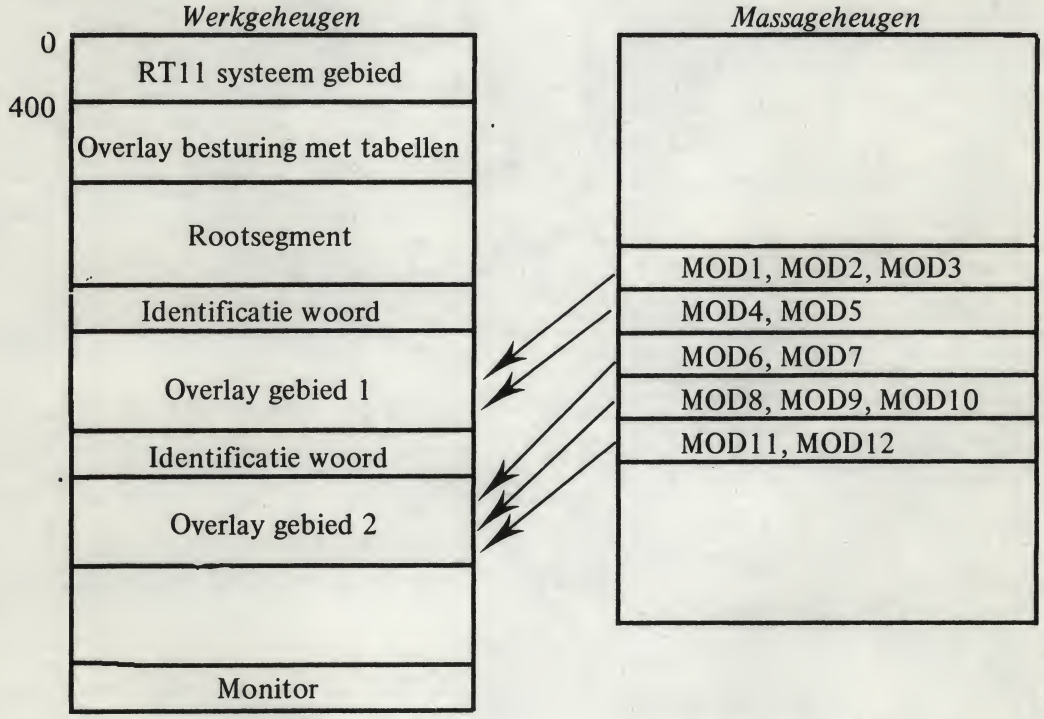


Gebruik van overlays

Indien een programma in overlay wordt gemaakt, bestaat dit programma uit een rootsegment, het deel dat permanent in het geheugen aanwezig is, en een overlay gebied met overlay segmenten die op een achtergrond geheugen staan.
Het rootsegment is dat deel van het programma, die niet "ge-overlayed" wordt. Hierin bevindt zich ondermeer informatie voor het verzorgen van de overlays.
De overlay segmenten worden m.b.v. link commando's toegewezen aan bepaalde overlay gebieden in het geheugen. De grootte van dit gebied wordt bepaald door het grootste overlay segment dat in dat gebied moet passen.
Een link fase, waarin van de overlay structuur gebruik maakt, kan er nu b.v. als volgt uitzien:

```
* OVLADE, TT: = ROOT.OBJ / C        (return)
* MOD1, MOD2, MOD3 / O : 1 / C      (return)
* MOD4, MOD5  / O : 1 / C           (return)
* MOD6, MOD7 / O :  2 / C           (return)
* MOD8, MOD9, MOD10 / O : 2 / C     (return)
* MOD11, MOD12 / O : 2              (return)
```

In dit voorbeeld zijn twee overlay gebieden gespecificeerd, resp. O : 1 en O : 2.
Het aantal overlay segmenten is vijf, twee voor overlay gebied 1 en drie voor overlay gebied 2. Als nu vanuit het rootsegment (ROOT.OBJ) MOD4 aangeroepen wordt, dan wordt overlaygebied 1 gevuld met MOD4 en MOD5, omdat deze samen een overlaysegment vormen.
De werk- en massageheugenopbouw voor dit programma is geschetst in fig. 18.



2° HET LIBR PROGRAMMA

Het LIBR programma voegt object modules samen in standaard bibliotheek vorm. Wanneer in de LINK fase een standaard bibliotheek gebruikt wordt, dan wordt uit zo'n standaard bibliotheek alleen die modules gepakt, waar gebruik van gemaakt wordt.
Aan de object files, die samen gevoegd moeten worden, moet een *module naam* toegekend zijn. Dit moet reeds in het assembler source programma met behulp van een .TITLE statement gebeuren.
Het LIBR programma wordt geladen en gestart met het volgende commando:

```
.R LIBR
```

Hierna moeten in- en output filespecificaties gegeven worden in de volgende gedaante:

```
*LIB, LIST = MOD1, MOD2, MOD3/S  (return)
```

Hierin is:

LIB	de bibliotheeknaam, die automatisch de extensie .OBJ krijgt toegekend.
,LIST	de filenaam of het apparaat waar de listing naar toe moet. De listing bevat module-namen en entry points.
MOD1, MOD2, MOD3	zijn de input filespecificaties (object modules)
/S	zijn functieschakelaars

b.v. /C commandostring vervolgt op volgende regel
/D verwijder de hierna te specificeren module
/G onderdruk entry point

Voorbeeld:

```
.R   LIBR
* LIBA, TT: = AAP, NOOT, MIES (return)
```

Samenvoegen van de objecten modules AAP, NOOT, MIES in de bibliotheek LIBA.

Voorbeeld:

```
.R   LIBR                    (return)
* LIBA = LIBA/D              (return)
Modulenaam
MIS (return)
(return)
*
```

Verwijder de module MIS uit de bibliotheek LIBA.

5.2.4. Hulpprogramma: ODT

Het ODT hulpprogramma kan gebruikt worden om fouten in een machinecodeprogramma op te sporen. Het programma is in vertaalde vorm, als object module aanwezig en kan met het te testen programma samengeknoopt worden. Het ODT programma kan gestart worden door het ontstane absolute programma met een GET commando in het geheugen te halen en te starten op de locatie O.ODT, te vinden in de linkmap.
Het ODT programma kent allerlei commando's om de inhoud van locaties te bekijken en te veranderen, delen van programma's te executeren etc. Een aantal commando's zijn in onderstaande lijst weergegeven.

RETURN	sluit geopende locatie
LINEFEED	sluit geopende locatie, open volgende locatie
n/	open locatie n
$n/	open algemeen register n
r; nB	plaats n^{o} breakpoint op locatie r
; nB	verwijder n^{o} breakpoint
;B	verwijder alle breakpoints
r; G	start op locatie r
; P	ga door na een breakpoint
k; P	ga door tot het breakpoint k keer bereikt is

Voor de volledige commandolijst wordt verwezen naar de RT11 system reference manual.

5.2.5. Tekstmanipulatie programma: EDIT

Voor de creatie en wijziging van ASCII tekstfiles, zoals assembler en fortran source programma's, bestaat het EDIT programma. Het EDIT programma werkt met een grote serie commando's, waarvan de belangrijkste beschreven zullen worden.
Het EDIT programma werkt met een tekstbuffer, waarin de tekst, eventueel een gedeelte van de tekst, indien met een z.g. pagina-indeling is gewerkt, wordt geplaatst.
In deze tekstbuffer wordt met een pointer gewerkt, die m.b.v. commando's door de tekst gemanoevreerd kan worden. Op de plaats, die de pointer aanwijst, kan tekst tussengevoegd, weggehaald en geprint worden.
Het EDIT programma wordt geladen en gestart met het commando: .R EDIT.
Hierna vraagt het EDIT programma om een filespecificatie. Hiervoor bestaan twee mogelijkheden:

1° Het maken van een nieuwe (nog niet) bestaande file:
* EWFILE .EXT (alt, alt)

Hierin is: EW de aanduiding dat het een nieuwe file betreft; → Edit WRITE
FILE.EXT een filespecificatie (zie par. 5.1.1)
(alt, alt) tweemaal drukken van de alt of altmode toets.

2° Het wijzigen van een bestaande file:
* EBFILE .EXT (alt, alt) → Edit Backup.

Hierin is: EB de aanduiding dat het een bestaande file betreft. Er ontstaat nu een nieuwe file met wijzigingen (FILE.EXT) terwijl tevens de oude, ongewijzigde file blijft bestaan onder de naam FILE.BAK.

De ALT of ALTMODE-toets wordt in het EDIT programma gebruikt als commando afsluiting. Dit in tegenstelling tot alle andere systeemprogramma's, die de RETURN of CR toets gebruiken als commando afsluiting.
De belangrijkste EDIT commando's zijn in onderstaande lijst weergegeven.

EDIT commando's:

EX	einde EDIT programma, terug naar monitor
R	leest een pagina tekst in de buffer
W	schrijft buffer in outputfile
N	schrijft buffer in outputfile en leest volgende pagina in buffer
nN	idem, maar (n - 1) keer
nL	print na pointer n regels op de terminal
V	print de regel waarin de pointer staat
B	plaatst de pointer aan het begin van de buffer
nJ	verplaatst de pointer n karakters
nA	verplaatst de pointer n regels
G tekst (alt)	plaatst de pointer achter de gespecificeerde tekst in tekstbuffer
F tekst (alt)	plaatst de pointer achter de gespecificeerde tekst in file
I tekst (alt)	voert de tekst in vanaf de pointer
nD	verwijder n karakters vanaf de pointer
nK	verwijder n regels vanaf de pointer
/	vanaf de pointer tot het eind van de buffer, b.v. /L

Het is toegestaan commando's samen te voegen tot een commandostring, b.v.:
* B / L (alt, alt)
Zoekt commando's G en F; zoeken vanaf de pointer.

Voorbeeld: In de volgende programma-tekst, die in de file NAAM.MAC aanwezig is, moet in regel 3 register R0 in R2 veranderd worden.

Programma:

```
START:    CLR   R0
          CLR   R1
          CLR   R0
          HALT
```

Het wijzigen gaat nu als volgt:

```
.R EDIT (return)                start EDIT programma
* EB NAAM.MAC (alt, alt)        wijzigt bestaande file NAAM.MAC
* R (alt, alt)                  leest file in tekstbuffer
* B (alt, alt)                  pointer naar het begin
* 2A (alt, alt)                 pointer voor derde regel
* GR0 (alt, alt)                plaatst de pointer achter R0
* -1 J (alt, alt)               plaatst de pointer 1 karakter terug
* DI 2 (alt, alt)               haalt karakter weg en plaatst 2
* V (alt, alt)                  controleert de gewijzigde regel
          CLR R2                o.k.
* EX (alt, alt)                 einde edit
```

5.2.6. File manipulatie programma: PIP

Het systeemprogramma PIP kan gebruikt worden voor het verplaatsen van files en voor het verrichten van systeem (software) onderhoud. Met PIP kunnen files van het ene naar het andere randapparaat gecopieerd worden, files kunnen samengevoegd of verwijderd worden. Verder kunnen directories (catalogi van files) geprint worden op een terminal. Commando's aan het PIP programma voldoen aan de standaard filespecificatie (zie par. 5.1.1.). Daarnaast bestaat de mogelijkheid om alle files met een bepaalde extensie, of alle files van een naam met alle extensies aan te duiden met het * teken:

```
*.MAC      alle files met extensie MAC.
TEST.*     alle files met naam TEST.
```

Naast de filespecificaties bestaan allerlei commando's, die in de vorm van switsches (/S) opgegeven worden.
Het PIP programma wordt geladen en gestart met het commando: .R PIP (return).
Hierna wordt om een filespecificatie(s) of een commando gevraagd. De algemene vorm is: * OUTPUT. EXT = INPUT.EXT /S

Voorbeeld:

```
.R PIP (return)
* CT0:  NAAM.MAC = NAAM.MAC
```

De file NAAM.MAC wordt vanaf het systeem medium gecopieerd naar de cassette unit 0 en hieronder dezelfde naam geplaatst.

Een aantal mogelijke switches (commando's) zijn:

/F print filenamen in gespecificeerde directory, b.v.: DT 1 : /F
/L print filenamen, afmetingen en verdere gegevens van gespecificeerde directory, b.v.: DT0: /L
/X copieert files één voor één, te gebruiken bij meervoudige copieropdrachten, b.v.: DT: * . * = DT0: * . MAC / X
/A bij copiëren van ASCII files
/R geeft nieuwe naam aan file, zonder copiëren, b.v.: NAAM.MAC =MIS.MAC/R (de file MIS.MAC krijgt nu de naam NAAM.MAC)
/D verwijdert gespecificeerde file, b.v.: NAAM.MAC/D
/C alleen betrekking op files met actuele datum.

Systeem (software) onderhoud commando's zijn niet in de lijst opgenomen omdat het foutief gebruik van deze commando's zeer snel kan leiden tot het "opblazen" van het software systeem. Dringend wordt aangeraden geen systeem onderhoud commando's te gebruiken zonder goedkeuring van de systeembeheerder.

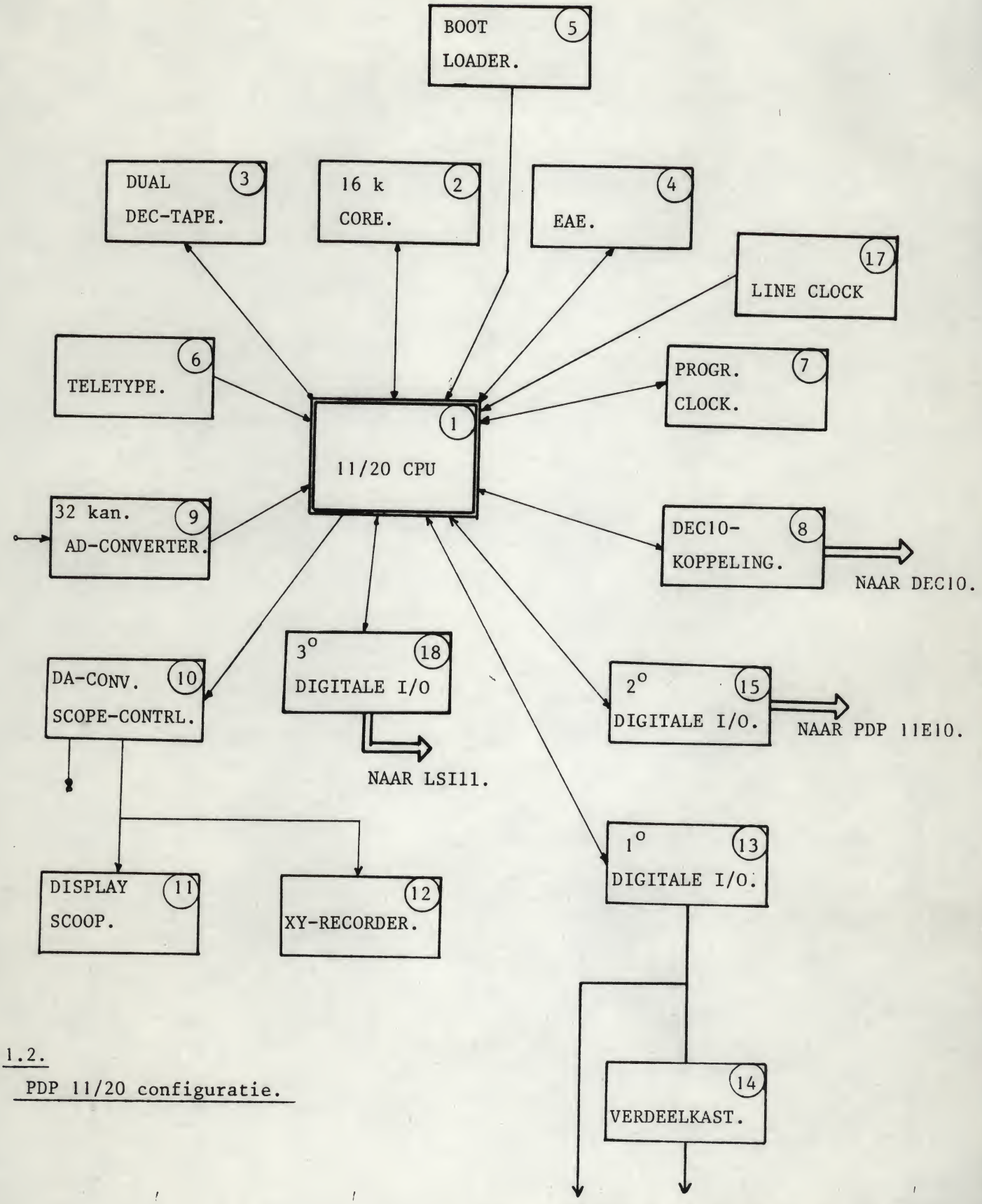
TECHNISCHE HOGESCHOOL TWENTE

APPENDIX A:

PDP-11 COMPUTERCONFIGURATIES van de vakgroep REGELTECHNIEK en AUTOMATISERING.

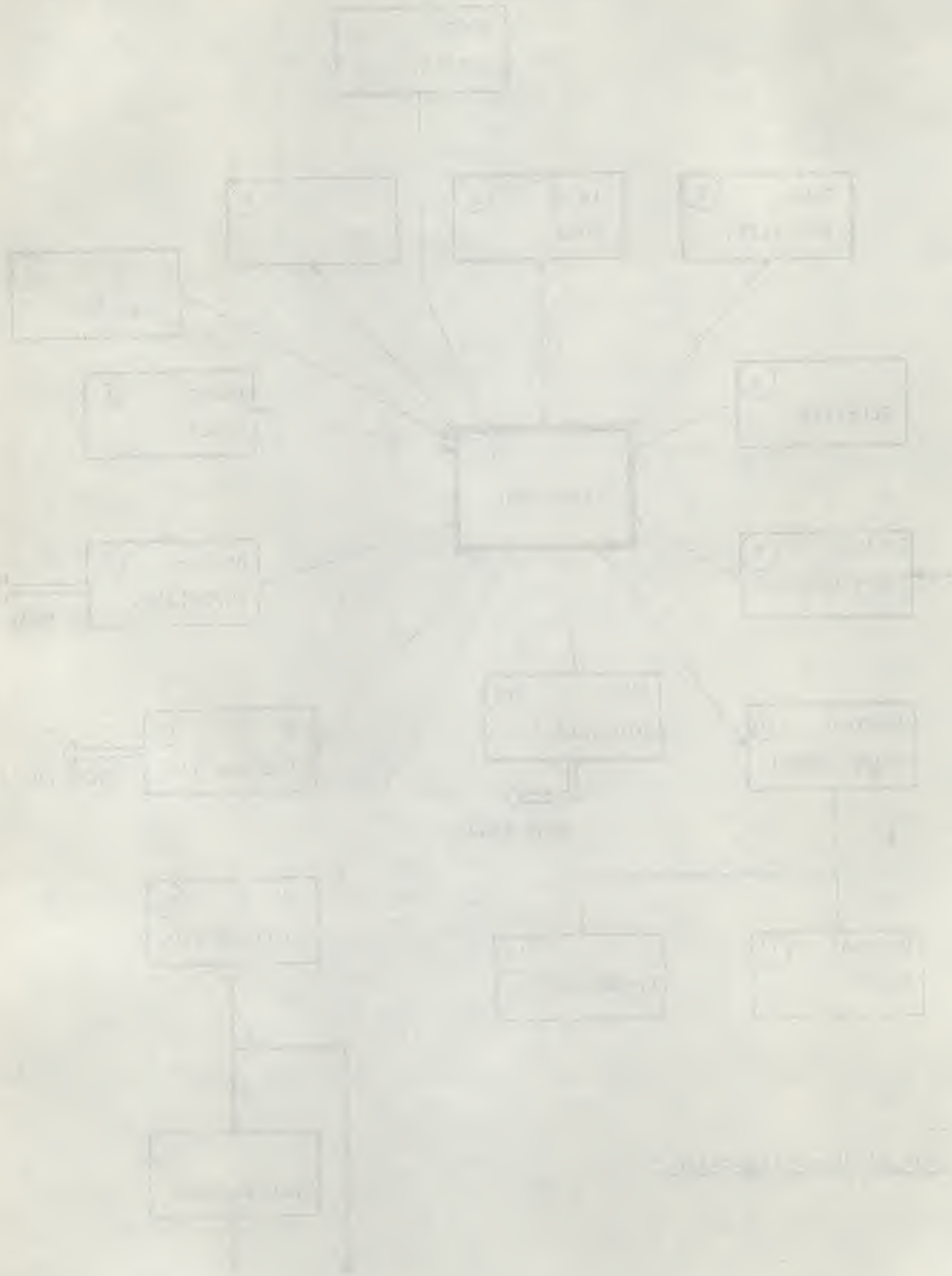
Inhoud:

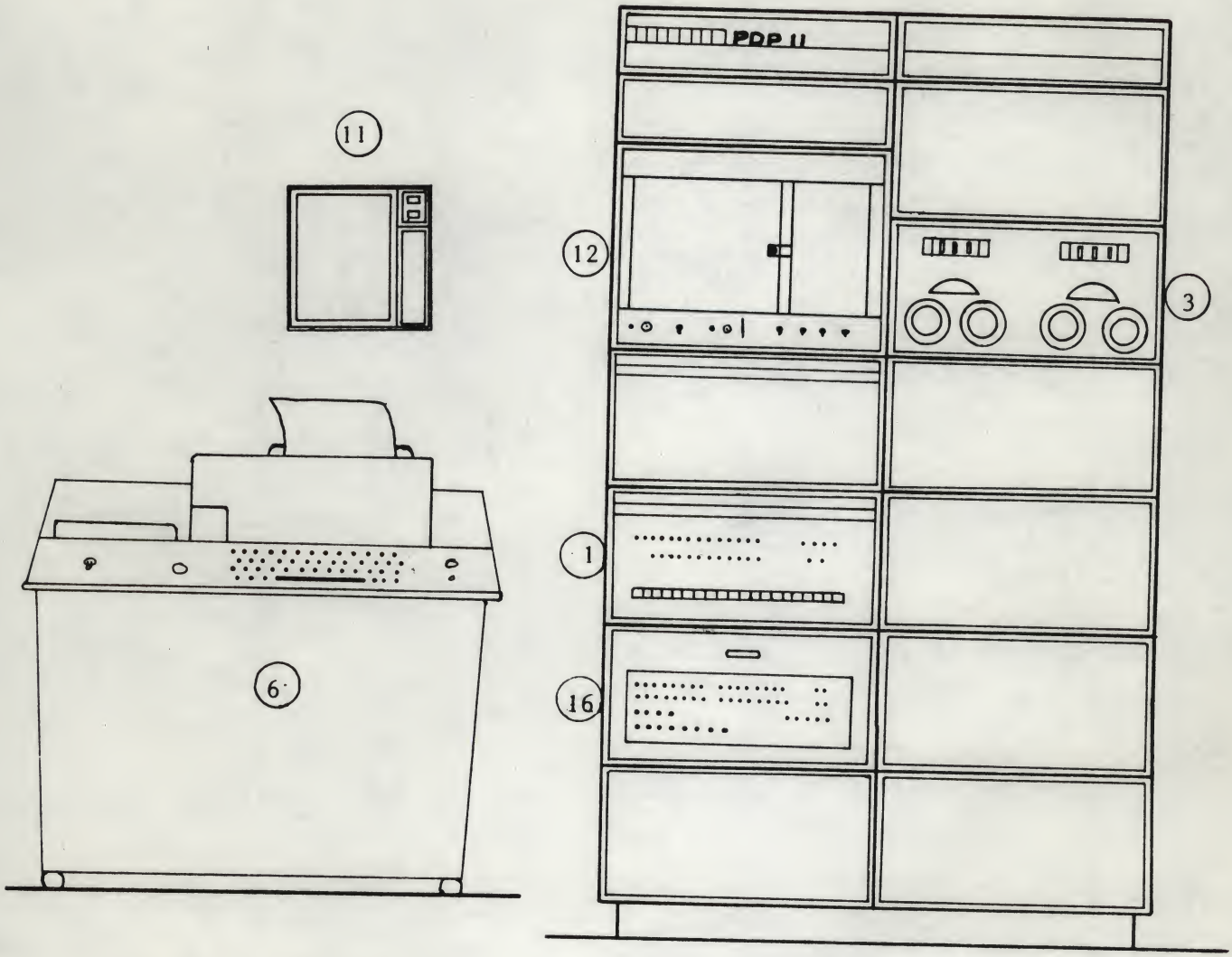
1.1	PDP 11/20 configuratie.	Blz. A2
1.2	PDP 11/20 opstelling.	Blz. A3
2.1	PDP 11E10 configuratie.	Blz. A5
2.2	PDP 11E10 opstelling.	Blz. A6
3.1	LSI 11/1 configuratie.	BLZ. A8
3.2	LSI 11/1 opstelling.	Blz. A8



1.2.

PDP 11/20 configuratie.





1.2 PDP 11/20 OPSTELLING.

TOELICHTING;

(1) = PDP 11/20 Centrale verwerkingseenheid. (CPU)
(2) = 16 k (woorden) kernengeheugen.
(3) = TC11 dubbele DEC-TAPE magneetbandunit.
(4) = KE11 hardware integer rekenunit.(EAE=Extended Arithmatic Element.)
(5) = BM792 bootstrap loader.
(6) = ASR35 Teletype.(Inclusief low speed papertape.)

(7) = KW11P Programmeerbare real time clock.

(8) = DL11C Serie lijninterface voor koppeling aan de DEC-10.

(9) = AD01 32 kanaals analoog/digitaal omzetter.(Met sign en S/H optie).

(10) = AA11 4 kanaals digitaal/analoog omzetter,inclusief AA11-A, scope-control voor 611-memoscope.

(11) = 611 Memoscope. (Display.)

(12) = 7041 XY-Recorder. (HP).

(13) = DR11C Digitale I/O. (Parallel interface voor algemene toepassingen).

(14) = Verdeelkast,om DR11C met meerdere toepassingen te verbinden.

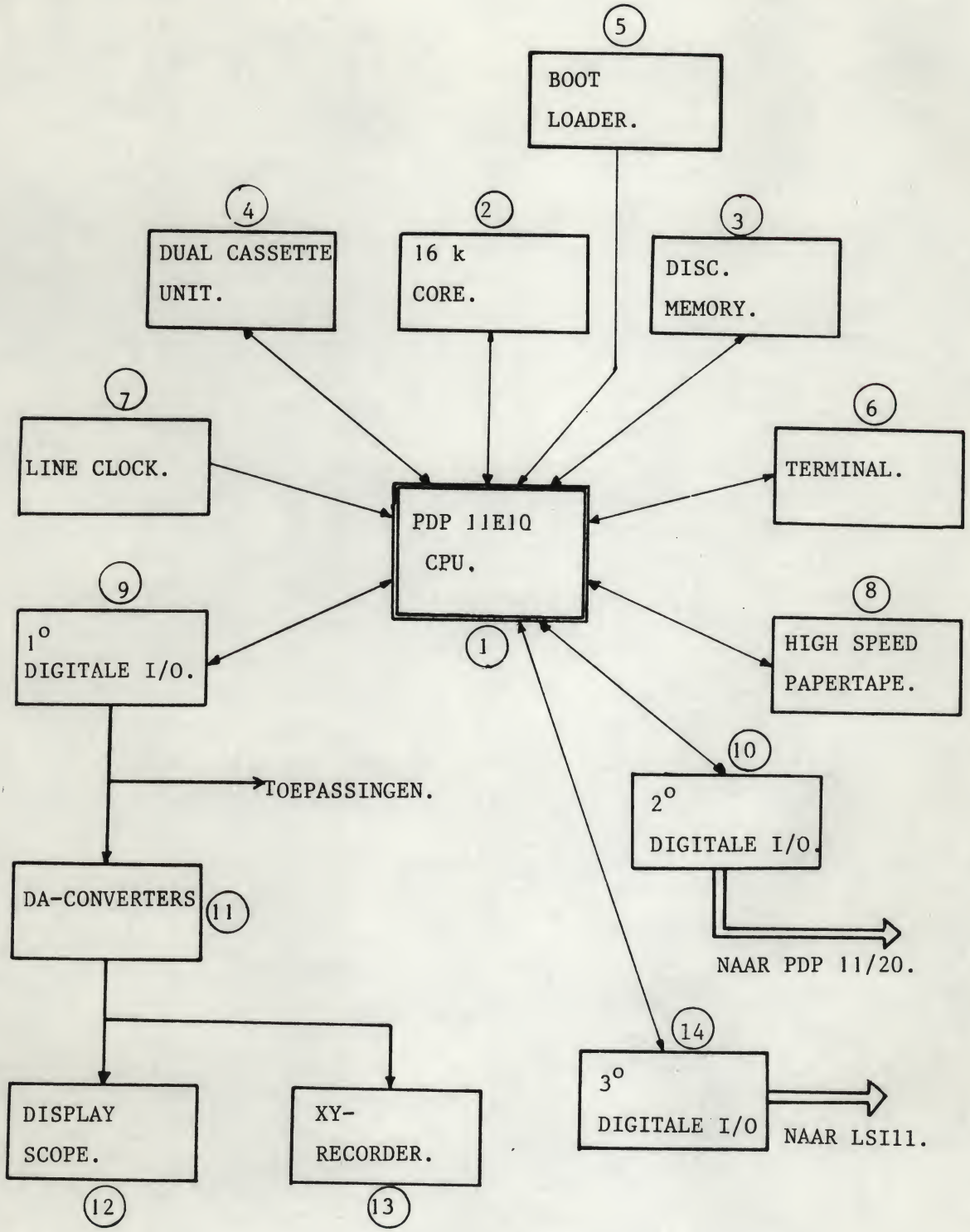
(15) = DR11C Digitale I/O. Parallel interface voor koppeling met PDP 11E10.

(16) = I/O paneel.

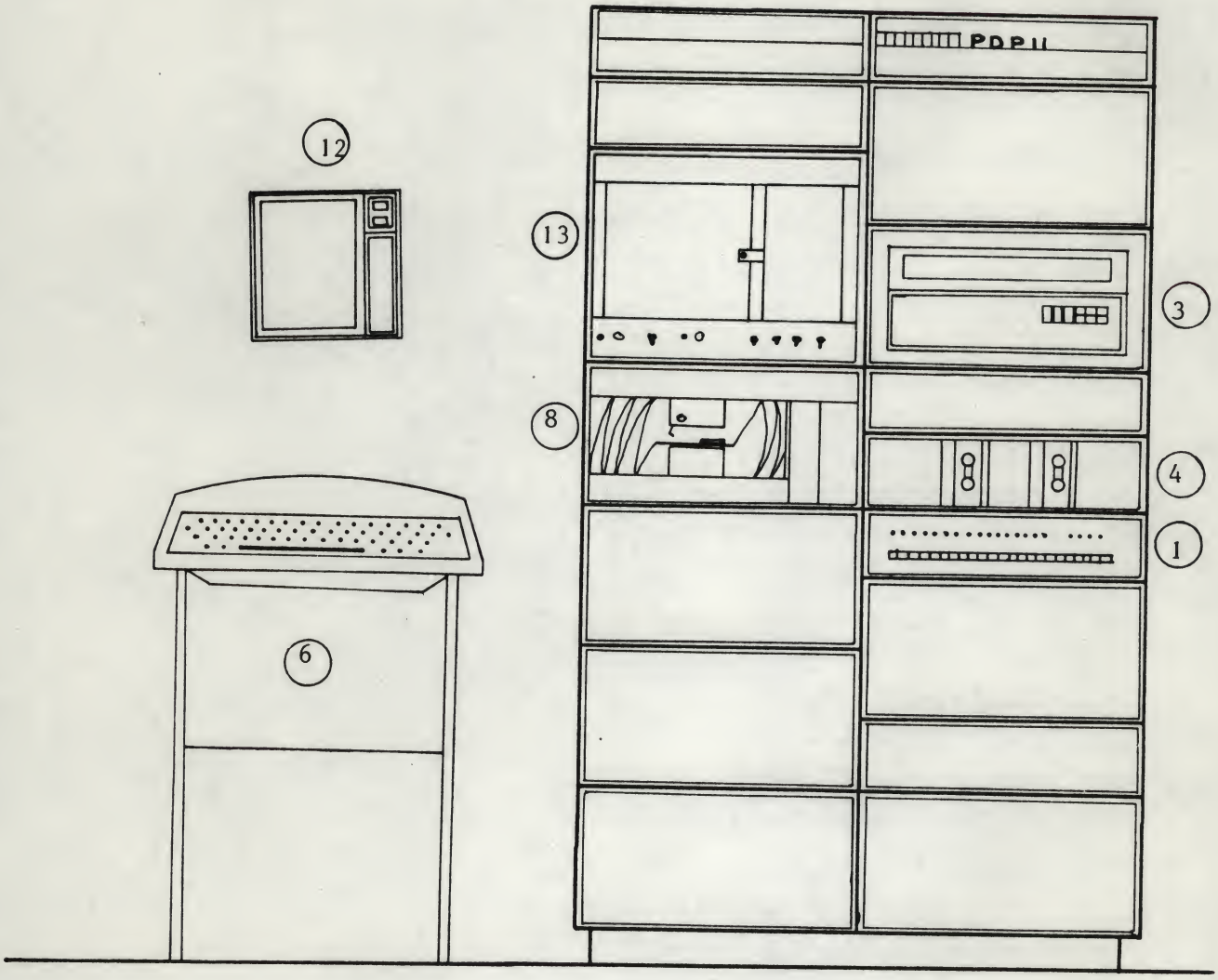
(17) = KW11L Netfrequentie clock.

(18) = DR11C Digitale I/O. Parallel-interface voor koppeling met LSI11.

N.B. Nadere gegevens van de DEC-(rand)apparatuur zijn te vinden in de DEC-uitgave: PDP11-peripherals handbook.



2.1 PDP 11E10 CONFIGURATIE.



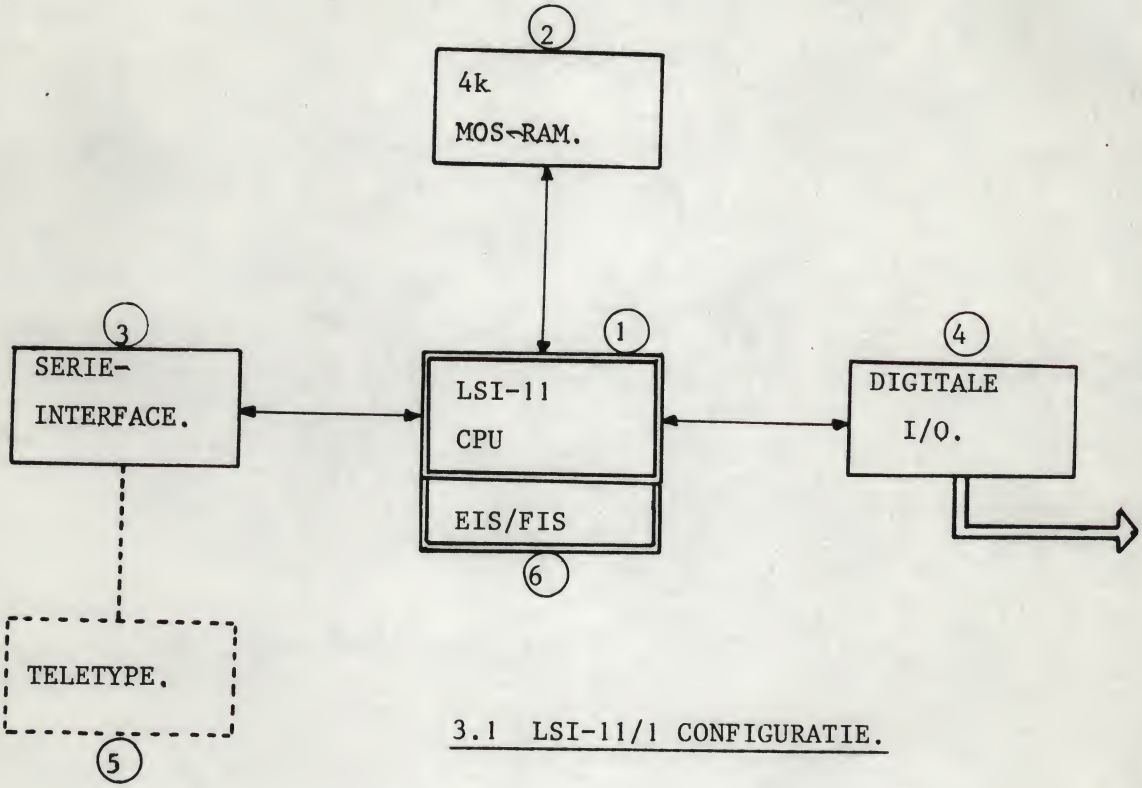
2.2 PDP 11E10 OPSTELLING.

TOELICHTING:

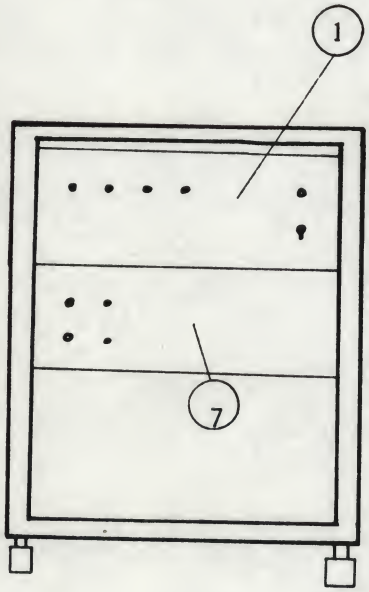
(1) = PDP 11E10 Centrale verwerkingseenheid.(CPU).
(2) = 16 k (woorden) kernengeheugen.
(3) = RK11 Schijfgeheugen.
(4) = TA11 Dubbele cassette unit.
(5) = BM792 Bootstrap loader.
(6) = LA36 terminal.

(7) = KW11L Netfrequentie clock.
(8) = PC11 High speed papertape punch & reader.
(9) = DR11C Digitale I/O. Parallel-interface naar eigenbouw DA-converters.
(Eventueel voor andere doeleinden te gebruiken).
(10) = DR11C Digitale I/O. Parallel-interface voor koppeling met PDP 11/20.
(11) = DA-converters,eigenbouw.
(12) = 611 Memory display scope.
(13) = 7041 XY-recorder. (HP).
(14) = DR11C Digitale I/O. Parallel-interface voor koppeling met LSI11.

N.B. Nadere gegevens van de DEC-(rand)apparatuur zijn te vinden in de DEC-uitgave: PDP11-peripherals handbook.



3.1 LSI-11/1 CONFIGURATIE.



3.2.
LSI-11 OPSTELLING.

TOELICHTING(LSI-11):

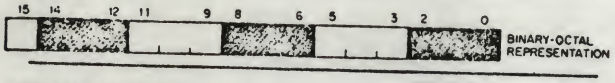
(1) = LSI-11 Centrale verwerkingseenheid.
(2) = 4 k (woorden) MOS-RAM geheugen.
(3) = DLV11 Serie lijninterface(Voor b.v. Teletype).
(4) = DRV11 Paralel interface voor koppeling met andere PDP-11 computers.
(5) = ASR33 Teletype. (Eventueel).
(6) = EIS/FIS floating point uitbreiding van de instructieset.

N.B. Nadere gegevens van de LSI-11 randapparatuur zijn te vinden in de LSI11,PDP11/03 Users manual.

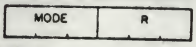
PROGRAMMING CARD

FOR FAMILY OF PDP-11 COMPUTERS

WORD FORMAT

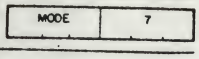


GENERAL REGISTER ADDRESSING



Mode	Name	Symbolic	Description
0	register	R	(R) is operand [ex. R2=%2]
1	register deferred	(R)	(R) is address
2	auto-increment	(R)+	(R) is adrs; (R) + (1 or 2)
3	auto-incr deferred	@(R)+	(R) is adrs of adrs; (R) + 2
4	auto-decrement	-(R)	(R) - (1 or 2); (R) is adrs
5	auto-decr deferred	@-(R)	(R) - 2; (R) is adrs of adrs
6	index	X(R)	(R) + X is adrs
7	index deferred	@X(R)	(R) + X is adrs of adrs

PROGRAM COUNTER ADDRESSING Reg = 7



2	immediate	#n	operand n follows instr
3	absolute	@#A	address A follows instr
6	relative	A	instr adrs + 4 + X is adrs
7	relative deferred	@A	instr adrs + 4 + X is adrs of adrs

LEGEND

Op Codes

■ = 0 for word/1 for byte
SS = source field (6 bits)
DD = destination field (6 bits)
R = gen register (3 bits), 0 to 7
XXX = offset (8 bits), +127 to -128
N = number (3 bits)
NN = number (6 bits)

Operations

( ) = contents of
s = contents of source
d = contents of destination
r = contents of register
← = becomes
X = relative address
% = register definition

Boolean

∧ = AND
∨ = inclusive OR
⊻ = exclusive OR
~ = NOT

Condition Codes

* = conditionally set/cleared
– = not affected
0 = cleared
1 = set

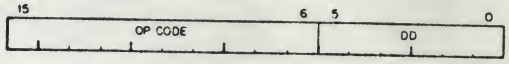
NOTE:

▲ = Applies to the 11/35, 11/40 & 11/45 computers
● = Applies to the 11/45 computer

7-BIT ASCII CODE

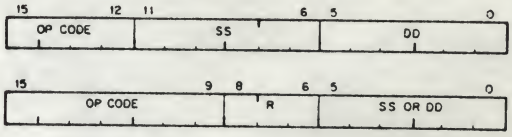
Octal Code	Char	Octal Code	Char	Octal Code	Char	Octal Code	Char
000	NUL	040	SP	100	@	140	`
001	SOH	041	!	101	A	141	a
002	STX	042	"	102	B	142	b
003	ETX	043	#	103	C	143	c
004	EOT	044	$	104	D	144	d
005	ENQ	045	%	105	E	145	e
006	ACK	046	&	106	F	146	f
007	BEL	047	'	107	G	147	g
010	BS	050	(	110	H	150	h
011	HT	051	)	111	I	151	i
012	LF	052	*	112	J	152	j
013	VT	053	+	113	K	153	k
014	FF	054	,	114	L	154	l
015	CR	055	-	115	M	155	m
016	SO	056	.	116	N	156	n
017	SI	057	/	117	O	157	o
020	DLE	060	0	120	P	160	p
021	DC1	061	1	121	Q	161	q
022	DC2	062	2	122	R	162	r
023	DC3	063	3	123	S	163	s
024	DC4	064	4	124	T	164	t
025	NAK	065	5	125	U	165	u
026	SYN	066	6	126	V	166	v
027	ETB	067	7	127	W	167	w
030	CAN	070	8	130	X	170	x
031	EM	071	9	131	Y	171	y
032	SUB	072	:	132	Z	172	z
033	ESC	073	;	133	[	173	{
034	FS	074	<	134	\	174	|
035	GS	075	=	135	]	175	}
036	RS	076	>	136	^	176	~
037	US	077	?	137	—	177	DEL

SINGLE OPERAND: OPR dst



Mnemonic	Op Code	Instruction	dst Result	N Z V C
General				
CLR(B)	■ 050DD	clear	0	0 1 0 0
COM(B)	■ 051DD	complement (1's)	~d	* * 0 1
INC(B)	■ 052DD	increment	d + 1	* * * –
DEC(B)	■ 053DD	decrement	d - 1	* * * –
NEG(B)	■ 054DD	negate (2's compl)	-d	* * * *
TST(B)	■ 057DD	test	d	* * 0 0
Rotate & Shift				
ROR(B)	■ 060DD	rotate right	→C, d	* * * *
ROL(B)	■ 061DD	rotate left	C, d←	* * * *
ASR(B)	■ 062DD	arith shift right	d/2	* * * *
ASL(B)	■ 063DD	arith shift left	2d	* * * *
SWAB	0003DD	swap bytes		* * * 0
Multiple Precision				
ADC(B)	■ 055DD	add carry	d + C	* * * *
SBC(B)	■ 056DD	subtract carry	d - C	* * * *
▲SXT	0067DD	sign extend	0 or -1	– * * –

DOUBLE OPERAND: OPR src, dst OPR src, R or OPR R, dst

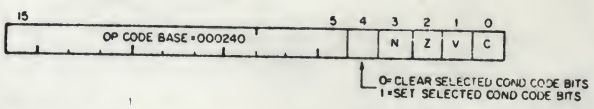


Mnemonic	Op Code	Instruction	Operation	N Z V C
General				
MOV(B)	■ 1SSDD	move	d ← s	* * 0 –
CMP(B)	■ 2SSDD	compare	s - d	* * * *
ADD	06SSDD	add	d ← s + d	* * * *
SUB	16SSDD	subtract	d ← d - s	* * * *
Logical				
BIT(B)	■ 3SSDD	bit test (AND)	s ∧ d	* * 0 –
BIC(B)	■ 4SSDD	bit clear	d ← (~s) ∧ d	* * 0 –
BIS(B)	■ 5SSDD	bit set (OR)	d ← s ∨ d	* * 0 –
▲Register				
MUL	070RSS	multiply	r ← r x s	* * 0 *
DIV	071RSS	divide	r ← r/s	* * * *
ASH	072RSS	shift arithmetically		* * * *
ASHC	073RSS	arith shift combined		* * * *
XOR	074RDD	exclusive OR	d ← r⊻d	* * 0 –

MISCELLANEOUS:

Mnemonic	Op Code	Instruction
HALT	000000	halt
WAIT	000001	wait for interrupt
RESET	000005	reset external bus
NOP	000240	(no operation)

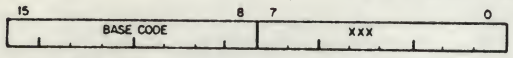
CONDITION CODE OPERATORS:



Mnemonic	Op Code	Instruction	N Z V C
CLC	000241	clear C	– – – 0
CLV	000242	clear V	– – 0 –
CLZ	000244	clear Z	– 0 – –
CLN	000250	clear N	0 – – –
CCC	000257	clear all cc bits	0 0 0 0
SEC	000261	set C	– – – 1
SEV	000262	set V	– – 1 –
SEZ	000264	set Z	– 1 – –
SEN	000270	set N	1 – – –
SCC	000277	set all cc bits	1 1 1 1

BRANCH: B – – location

If condition is satisfied:
Branch to location,
New PC ← Updated PC + (2 x offset)
adrs of br instr + 2



Op Code = Base Code + XXX

Mnemonic	Base Code	Instruction	Branch Condition	
Branches				
BR	000400	branch (unconditional)	(always)	
BNE	001000	br if not equal (to 0)	≠ 0	Z = 0
BEQ	001400	br if equal (to 0)	= 0	Z = 1
BPL	100000	branch if plus	+	N = 0
BMI	100400	branch if minus	-	N = 1
BVC	102000	br if overflow is clear		V = 0
BVS	102400	br if overflow is set		V = 1
BCC	103000	br if carry is clear		C = 0
BCS	103400	br if carry is set		C = 1
Signed Conditional Branches				
BGE	002000	br if greater or eq (to 0)	≥0	N⊻V = 0
BLT	002400	br if less than (0)	<0	N⊻V = 1
BGT	003000	br if greater than (0)	>0	Z v (N⊻V) = 0
BLE	003400	br if less or equal (to 0)	≤0	Z v (N⊻V) = 1
Unsigned Conditional Branches				
BHI	101000	branch if higher	>	C v Z = 0
BLOS	101400	branch if lower or same	≤	C v Z = 1
BHIS	103000	branch if higher or same	≥	C = 0
BLO	103400	branch if lower	<	C = 1

JUMP & SUBROUTINE:

Mnemonic	Op Code	Instruction	Notes
JMP	0001DD	jump	PC ← dst
JSR	004RDD	jump to subroutine	use same R
RTS	00020R	return from subroutine	use same R
▲MARK	0064NN	mark	aid in subr return
▲SOB	077RNN	subtract 1 & br (if ≠ 0)	(R) – 1, then if (R) ≠ 0: PC ← Updated PC – (2 x NN)

TRAP & INTERRUPT:

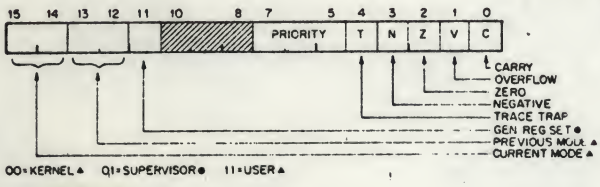
Mnemonic	Op Code	Instruction	Notes
EMT	104000 to 104377	emulator trap (not for general use)	PC at 30, PS at 32
TRAP	104400 to 104777	trap	PC at 34, PS at 36
BPT	000003	breakpoint trap	PC at 14, PS at 16
IOT	000004	input/output trap	PC at 20, PS at 22
RTI	000002	return from interrupt	
▲RTT	000006	return from interrupt	inhibit T bit trap

PDP-11/40 FLOATING POINT UNIT:

			N	Z	V	C
FADD	07500R	floating add	*	*	0	0
FSUB	07501R	floating subtract	*	*	0	0
FMUL	07502R	floating multiply	*	*	0	0
FDIV	07503R	floating divide	*	*	0	0

PROCESSOR REGISTER ADDRESSES

Processor Status Word
PS - 777 776



▲Stack Limit Register — 777 774

●Program Interrupt Request —777 772

General Registers (console use only)	R0 — 777 700	R4 — 777 704
	R1 — 777 701	R5 — 777 705
	R2 — 777 702	R6 — 777 706
(not for 11/45)	R3 — 777 703	R7 — 777 707

Console Switches & Display Register — 777 570

NUMERICAL OP CODE LIST

OP Code	Mnemonic	OP Code	Mnemonic	OP Code	Mnemonic
00 00 00	HALT	00 60 DD	ROR	10 40 00 ↕ 10 43 77	EMT
00 00 01	WAIT	00 61 DD	ROL		
00 00 02	RTI	00 62 DD	ASR		
00 00 03	BPT	00 63 DD	ASL		
00 00 04	IOT	00 64 NN	MARK	10 44 00 ↕ 10 47 77	TRAP
00 00 05	RESET	00 65 SS	MFPI		
00 00 06	RTT	00 66 DD	MTPI		
00 00 07 ↕ 00 00 77	(unused)	00 67 DD	SXT		
		00 70 00 ↕ 00 77 77	(unused)	10 50 DD	CLRB
00 01 DD	JMP			10 51 DD	COMB
00 02 0R	RTS			10 52 DD	INCB
		01 SS DD	MOV	10 53 DD	DECB
00 02 10 ↕ 00 02 27	(unused)	02 SS DD	CMP	10 54 DD	NEGB
		03 SS DD	BIT	10 55 DD	ADCB
		04 SS DD	BIC	10 56 DD	SBCB
00 02 3N	SPL	05 SS DD	BIS	10 57 DD	TSTB
00 02 40	NOP	06 SS DD	ADD		
				10 60 DD	RORB
00 02 41 ↕ 00 02 77	cond codes	07 0R SS	MUL	10 61 DD	ROLB
		07 1R SS	DIV	10 62 DD	ASRB
		07 2R SS	ASH	10 63 DD	ASLB
		07 3R SS	ASHC		
00 03 DD	SWAB	07 4R DD	XOR	10 64 00 ↕ 10 64 77	(unused)
		07 50 0R	FADD		
00 04 XXX	BR	07 50 1R	FSUB		
00 10 XXX	BNE	07 50 2R	FMUL	10 65 SS	MFPD
00 14 XXX	BEQ	07 50 3R	FDIV	10 66 DD	MTPD
00 20 XXX	BGE				
00 24 XXX	BLT	07 50 40 ↕ 07 67 77	(unused)	10 67 00 ↕ 10 77 77	(unused)
00 30 XXX	BGT				
00 34 XXX	BLE				
		07 7R NN	SOB		
00 4R DD	JSR				
		10 00 XXX	BPL	11 SS DD	MOVB
00 50 DD	CLR	10 04 XXX	BMI	12 SS DD	CMPB
00 51 DD	COM	10 10 XXX	BHI	13 SS DD	BITB
00 52 DD	INC	10 14 XXX	BLOS	14 SS DD	BICB
00 53 DD	DEC	10 20 XXX	BVC	15 SS DD	BISB
00 54 DD	NEG	10 24 XXX	BVS	16 SS DD	SUB
00 55 DD	ADC	10 30 XXX	BCC, BHIS	17 00 00 ↕ 17 77 77	floating point
00 56 DD	SBC	10 34 XXX	BCS, BLO		
00 57 DD	TST				

TRAP VECTORS

000	(reserved)	114	Memory Parity
004	Time Out & other errors	240	PIRQ, prog int req
010	illegal & reserved instr	244	Floating Point
014	BPT instruction	250	Memory Management
020	IOT instruction		
024	Power Fail		
030	EMT instruction		
034	TRAP instruction		

DEVICE REGISTER ADDRESSES

Device	Registers	Address	Int Vec-tor	Prior-ity	NPR
KW11-L Line Clock	(LKS)	777 546	100	BR6	
KW11-P Programmable Clock			104	BR6	
control & status		772 540			
count set buffer		772 542			
counter		772 544			
LA30, Console Terminal					
LT33, keyboard/reader status		777 560	60	BR4	
VT05 keyboard/reader buffer		777 562			
printer/punch status		777 564	64	BR4	
printer/punch buffer		777 566			
PC11 Paper Tape					
reader status	(PRS)	777 550	70	BR4	
reader buffer	(PRB)	777 552			
punch status	(PPS)	777 554	74	BR4	
punch buffer	(PPB)	777 556			
RK11/ Disk Cartridge			220	BR5	X
RK05 drive status	(RKDS)	777 400			
error	(RKER)	777 402			
control & status	(RKCS)	777 404			
word count	(RKWC)	777 406			
current bus adrs	(RKBA)	777 410			
disk address	(RKDA)	777 412			
data buffer	(RKDB)	777 416			
TA11 Cassette			260	BR6	
command & status	(TACS)	777 500			
data buffer	(TADB)	777 502			
TC11/ DECtape			214	BR6	X
TU56 control & status	(TCST)	777 340			
command	(TCCM)	777 342			
word count	(TCWC)	777 344			
bus address	(TCBA)	777 346			
data	(TCDT)	777 350			

APPENDIX C: FORTRAN op de PDP11.

Bij het maken van een FORTRAN programma moeten een aantal fasen doorlopen worden. In fig. C1 is dit in een blokdiagram weergegeven.

Als eerste fase kan onderscheiden worden het schrijven van het programma. Hiervoor is de FORTRAN programmeertaal ter beschikking, waarop hier echter niet verder ingegaan wordt, omdat in de literatuur vele FORTRAN taalbeschrijvingen beschikbaar zijn. Goed aansluitend bij de toepassing op de PDP11 is de manual van Digital:

PDP11 FORTRAN LANGUAGE REFERENCE MANUAL

verkrijgbaar bij het rekencentrum.

In de tweede fase moet het geschreven FORTRAN programma in de computer (op massageheugen) ingevoerd worden. Dit gebeurt met het tekstmanipulatieprogramma EDIT. Het EDIT programma is in hoofdstuk 5 van deze handleiding beschreven. Met het EDIT programma wordt de programmatekst ingevoerd, gewijzigd en als file op het massageheugen geplaatst. Een file is een stuk informatie op massageheugen waaraan een unieke naam is toegekend.

In de derde fase wordt het op massageheugen aanwezige FORTRAN programma met behulp van het FORTRAN vertaal programma vertaald. Er ontstaat na de vertaling een object module (ook weer op massageheugen), die nog met de FORTRAN bibliotheek en eventuele andere programma's samengeknoopt moet worden. Dit samenknopen gebeurt in de vierde fase met behulp van het programma LINK.

De gang van zaken bij het vertalen van een FORTRAN programma is hieronder weergegeven. De te verrichten handelingen zijn achtereenvolgens:

Type na het RT11 .teken: .R FORTRAN

Type na het RT11 *teken: *OUTPUT,LIST=INPUT/S

b.v.: *DEMO,TT:=DEMO/L:0

HIERIN IS:

(zie pag. C3)

BLOKDIAGRAM: FORTRAN PROGRAMMERING OP DE PDP11

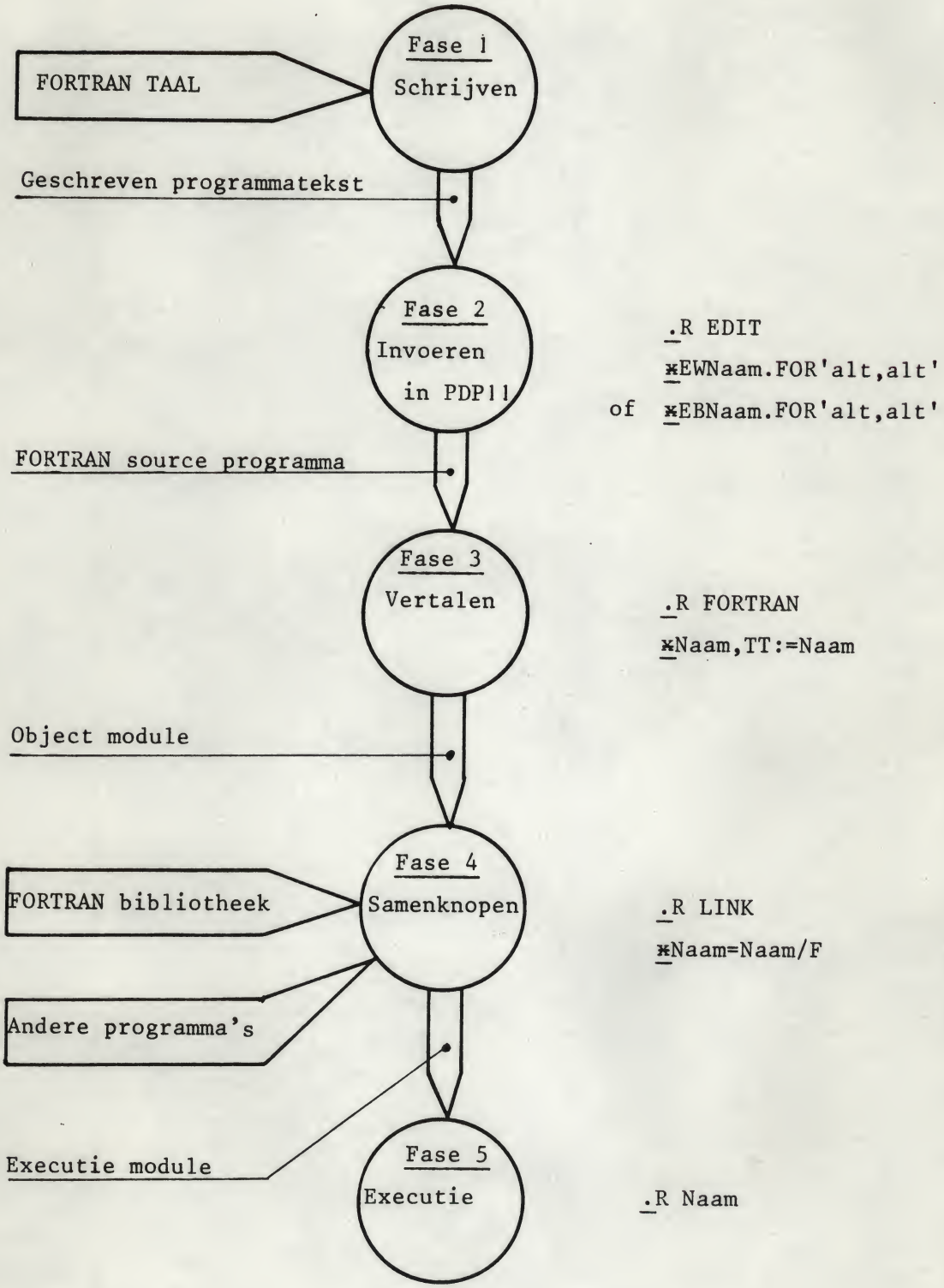


Fig. C1. Blokdiagram FORTRAN programmering PDP11.

OUTPUT	- file specificatie van het output object module.
LIST	- apparaat specificatie voor de listing, b.v. TT:
INPUT	- file specificatie van het te vertalen FORTRAN programma.
/S	- z.g. switches, met de volgende betekenis:
/H	Druk een lijst met gebruikte switches af.
/L:n	Specificatie listing opties:
	n=0 list alleen boodschappen.
	n=1 list alleen het source programma.
	n=2 list het geheugengebruik (storage map).
	n=4 list de gegenereerde code.
/S	Onderdruk interne statement nummers.
/T	Standaard lengte integers wordt twee woorden.
/W	Sta compiler waarschuwingen toe.

Nadat de vertaling heeft plaatsgevonden kan het programma samengeknoopt worden. Dit gebeurt met het programma LINK. Ook dit programma is in hoofdstuk 5 beschreven.
De procedure is de volgende:

. R LINK
* OUTPUT=INPUT/F

Hierin is:

OUTPUT	-file specificatie van het output laadprogramma.
INPUT	-file specificatie van het input object programma (Eventueel meerdere).
/F	-geeft aan dat het een FORTRAN programma betreft en dat de FORTRAN bibliotheek gebruikt moet worden.

Een aantal mogelijkheden die het LINKprogramma biedt zijn:

- Rangschikken objektmodulen en toekennen geheugenruimte.
- Verbindingen leggen tussen modulen met overeenkomstige globale symbolen.
- Verzorging van overlay strukturen.
- Geeft eventueel een load map, waarin te zien is waar de modulus en globale symbolen in het geheugen terecht komen.
- Zoekt gespecificeerde bibliotheken af naar nog ontbrekende symbolen.
- Plaatst het eindprodukt, een z.g. load module, op massageheugen.

Nadat met het LINK programma een load module gemaakt is, is het programma klaar voor executie. Het programma kan geëxecuteerd worden met een RUN commando, b.v.:

. R PROG

waarin PROG de naam is die aan het load module is toegekend.

Voorbeeld:

Een FORTRAN programma is reeds op het massageheugen aanwezig onder de naam: DEMO.FOR

Dit programma berekent en print na het opgeven van de beginvoorwaarden een z.g. Fibonaccireeks.

De gang van zaken, om dit onvertaalde programma aan de praat te krijgen is hieronder weergegeven, waarbij alles dat door de computer geprint is onderstreept is.

Start RT11. ('CR'=RETURN of CR toets).

RT11.....

. R FORTRAN 'CR'

* DEMO,TT:=DEMO'CR'

:

listing

:

* 'CNTL/C' (gelijktijdig 'CNTL' en 'C' toets)

. R LINK'CR'

* DEMO=DEMO/F'CR'

* 'CNTL/C'

. R DEMO

:

programma gaat werken

:

Enige aanvullende opmerkingen.

Bij het schrijven van FORTRAN programma's is het handig uit te gaan van FORTRAN coding sheets. Dit zijn papieren, waarop een juiste indeling gemaakt is. Het programma kan op dezelfde manier als op de codingsheets ingevoerd worden. Dit heeft echter het bezwaar dat de meeste statements dan met een aantal spaties beginnen, die steeds

uitgeteld moeten worden. Daarom is het handiger gebruik te maken van tabulaties. Enige voorbeelden zijn:

```
→|1ΔHOLD,MOVE      komt overeen met ΔΔΔΔΔ1ΔHOLD,MOV
1O→|X(I)=28        komt overeen met 1OΔΔΔΔX(I)=28
→|CONTINUE         komt overeen met ΔΔΔΔΔΔCONTINUE
```

waarin:→| een tabulatiekarakter aangeeft
Δ een spatie voorstelt

Data formaten.

Integers:
- Standaard : één 16 bits woord
- Bij gebruik /T switch wordt dubbel precisie formaat gereserveerd, doch de berekeningen maken alleen gebruik van het deel met het laagste adres.

Reals:
- Standaard: 2 woords floating point.
- Dubbele precisie: 4 woords floating point
- Complex: twee keer 2 woords floating point
  laagste adres - reële deel
  hoogste adres - imaginair deel

SYSLIB.

Door gebruik te maken van SYSLIB, de RT11-FORTRAN systeem subroutine bibliotheek, komen alle RT11 monitor mogelijkheden en diensten ter beschikking van de FORTRAN programmeur.
FORTRAN programma's kunnen subroutines zo indelen, dat ze geëxecuteerd worden als een van buiten komende gebeurtenis plaatsvindt. (Afgelopen tijdsinterval, completeren randapparaat actie). Het verrichten van allerlei soorten input/output en systeem informatie aanvragen op RT11 monitor niveau behoort tot de mogelijkheden, evenals de behandeling van interrupts met FORTRAN subroutine.
Daarnaast geeft SYSLIB de mogelijkheid tot uitgebreide string manipulaties.

Logical Unit numbers.

FORTRAN maakt o.a. bij het READ en WRITE statement gebruik van z.g. "logical unit numbers", om te specificeren op welk randapparaat het betreffende statement betrekking heeft. Enige veel voorkomende nummers

zijn:

apparaat		(zie Note) logical unit number
System unit	SY:	1
Schijf unit	DK:	2,3,4
Terminal (input)	TT:	5
Terminal(output)	TT:	7

Aan een apparaat kan m.b.v. een monitor commando ASSIGN een logical unit number toegekend worden.

Koppeling assembler programma's aan een FORTRAN programma.

Om assembler programma's (subroutines) binnen een FORTRAN programma te kunnen gebruiken moet met een aantal zaken rekening gehouden worden.

1e. Uit de assembler subroutine moet teruggekeerd worden met: RTS PC

2e. Overdracht van variabelen vindt plaats met behulp van algemeen register R5. Bij de aanroep vanuit het FORTRAN programma wijst R5 naar een gebied waar achtereenvolgens geplaatst zijn:

1e. aantal variabelen, dat bij de aanroep betrokken is.

2e. de adressen van de bij de aanroep betrokken variabelen.

In onderstaande schets is dit in beeld gebracht.

```
R5 ⟹  aantal variabelen
      adres 1e variabele
      adres 2e variabele
            etc
```

3e. In het FORTRAN programma wordt de assembler subroutine aangeroepen met een CALL statement, b.v.: CALL ADCONV(CHAN)

In de linkfase moet ook het assembler programma met het FORTRAN programma samengeknoopt worden.

Note: Bij gebruik van files op massageheugen (zoals SY:) gelden de volgende voorgeschreven file-namen:

Bij invoer ⟶ FTN1.DAT

Bij uitvoer ⟶ FTN2.DAT

Overzicht FORTRAN RT11.

FORTRAN IV

Statements

Arithmetic or Logical Assignment
a = b
Assigns value of expression b to variable a

Arithmetic Statement Function
tname (a1,...)=x
Creates user-defined function having variables a1,... as dummy arguments

ACCEPT f, list
Reads input from logical unit 5 (default TT:); f is format statement label, and list is optional data list

ASSIGN n TO ivar
Assigns statement number n to integer variable ivar

BACKSPACE u
Currently open file on logical unit number u is backspaced one record

BLOCK DATA
Specifies subprogram which follows as BLOCK DATA subprogram

CALL {name / name (arg1, arg2,...,)}
Calls SUBROUTINE subprogram with name specified, passing actual arguments (arg) to replace dummy arguments in SUBROUTINE definition

COMMON/name1/var1,.../name2/var,var
Reserves one or more blocks of storage space under name specified to contain variables (var) associated with that block name

CONTINUE
Causes no processing

DATA var1, var2,.../val1,val2,.../
Causes elements in list of values (val) to be initially stored in corresponding elements of list of variable names

DECODE (c,f,v) list
Changes elements in list of variables from ASCII into desired internal format (c is number of characters, f is format specifier, v is array name)

DEFINE FILE u (m,n,U,ivar),...
Defines record structure of direct access file

DIMENSION s1, s2,...,sk
Reserves storage space for specified arrays

DO n ivar = e1,e2,e3
1. Sets integer variable (ivar) = to the expression (e1)
2. Executes statements through statement number n
3. Increments ivar = ivar+e3
4. If e3 > 0 and ivar < = e2, or e3 < 0 and ivar > = e2, goes back to 2 above, else falls through to statement following statement n

END
Delimits a program unit

ENCODE (c,f,v) list
Changes elements in list of variables into ASCII format (c is number of characters, f is format specifier, v is array name)

END FILE u
Closes file currently open on logical unit number u

EQUIVALENCE (var1, var2,...), (varj,vark,...)
Assigns same storage location to each variable within set of parentheses

EXTERNAL name1, name2,...
Informs system that names specified are those of FUNCTION or SUBROUTINE subprograms

FIND (u'r)
Positions direct access file on logical unit number u to record r and sets associated variable to record number r

FORMAT (field specification,...)
Describes format in which one or more records are to be transmitted

FUNCTION name
FUNCTION name (var1, var2,...)
type FUNCTION name (var1, var2,...)
Begins FUNCTION subprogram, indicating program name and any dummy variable names (var); optional type specification can be included

Unconditional GOTO
GOTO n
Transfers control to statement number n

Computed GOTO
GOTO (k1, k2,...kn), e
Transfers control to statement number ki where i = value of expression e. If e < 1 or e > n, no transfer takes place.

Assigned GOTO
GOTO ivar
GOTO ivar, (k1, k2,...kn)
Transfers control to statement most recently associated with ivar by an ASSIGN statement

Arithmetic IF
IF (expression) n1,n2,n3
Transfers control to statement number n depending upon value of the expression

Logical IF
IF (expression) statement
Executes statement if logical expression tests true

IMPLICIT type (a1,b2-b3,...),...
Elements a and b represent single (or a range of) letter(s) whose presence as the initial letter of a variable specifies variable to be of that type, if that variable is not explicitly given a type

PAUSE
PAUSE display
Suspends program execution and prints octal constant, decimal constant, or alphanumeric literal display, if one is specified; resume execution by typing carriage return

PRINT f, list
Writes output on logical unit 6 (default LP:); f is format statement label, list is optional data list

Formatted READ
READ(u,f)
READ(u,f) list
READ f,list
Reads at least one logical record from device u (default 1) according to format specification f and assigns values to the variables in the list

Unformatted READ
READ(u)
READ(u) list
Reads one logical record from device u, assigning values to variables in list

Direct Access READ
READ(u'r) list
Reads from logical unit number u, record number r, and assigns values to variables in list

Transfer of Control on Error
END=n
ERR=m
END=n,ERR=m
Optional elements in READ statement list allowing control transfer on end of file conditions or error

RETURN
Returns control to calling program from current subprogram

REWIND u
Repositions logical unit number u to beginning of currently opened file

STOP
STOP display
Terminates program execution and prints octal constant, decimal constant, or alphanumeric literal display, if one is specified

SUBROUTINE name
SUBROUTINE name (var1,var2,...)
Begins a SUBROUTINE subprogram, indicating program name and any dummy variable names (var)

TYPE f,list
Writes output on logical unit 7 (default TT:); f is format statement label and list is optional data list

Type Declarations
type var1,var2,...vark
Variable names (var) are assigned specified data type in program unit; type = REAL, INTEGER, DOUBLE PRECISION, LOGICAL, or COMPLEX. An optional byte length may be given after type or after any variable name (e.g., INTEGER * 2).

Formatted WRITE
WRITE (u,f)
WRITE (u,f) list
Causes one or more logical records containing values of variables in list to be written onto logical unit u according to format specification f

Unformatted WRITE
WRITE (u)
WRITE (u) list
Causes one or more logical records containing values of variables in list to be written onto logical unit u

Direct Access WRITE
WRITE (u'r) list
Causes one logical record containing values of variables in list to be written onto record r of logical unit number u

Transfer of Control on Error
END=n
ERR=m
END=n,ERR=m
Optional elements in WRITE statement list allowing control transfer on end-of-file conditions or error

Library Functions

(arg type → result type in all cases)

ABS(X)	Real absolute value	Real → Real
IABS(I)	Integer absolute value	Integer → Integer
DABS(X)	Double precision absolute value	Double → Double
CABS(Z)	Complex to Real, absolute value where $Z=(x,y)$ $CABS(Z)=(x^2+y^2)^{1/2}$	Complex → Real
FLOAT(I)	Integer to Real conversion	Integer → Real
IFIX(X)	Real to Integer conversion IFIX(X) is equivalent to INT(X)	Real → Integer
SNGL(X)	Double to Real conversion	Double → Real
DBLE(X)	Real to Double conversion	Real → Double
REAL(Z)	Complex to Real conversion, obtain real part	Complex → Real
AIMAG(Z)	Complex to Real conversion, obtain imaginary part	Complex → Real
CMPLX(X,Y)	Real to Complex conversion CMPLX(X,Y)=X+i*Y	Real → Complex

Truncation functions return the sign of the argument * largest integer ⩽ | arg |

AINT(X)	Real to Real truncation	Real → Real
INT(X)	Real to Integer truncation	Real → Integer
IDINT(X)	Double to Integer truncation	Double → Integer

Remainder functions return the remainder when the first argument is divided by the second

AMOD(X,Y)	Real remainder	Real → Real
MOD(I,J)	Integer remainder	Integer → Integer
DMOD(X,Y)	Double precision remainder	Double → Double

Maximum value functions return the largest value from among the argument list; ⩾ 2 arguments

AMAX0(I,J,...)	Real maximum from Integer list	Integer → Real
AMAX1(X,Y,...)	Real maximum from Real list	Real → Real
MAX0(I,J,...)	Integer maximum from Integer list	Integer → Integer
MAX1(X,Y,...)	Integer maximum from Real list	Real → Integer
DMAX1(X,Y,...)	Double maximum from Double list	Double → Double

Library Functions (continued)

Minimum value functions return the smallest value from among the argument list; ⩾ 2 arguments

AMIN0(I,J,...)	Real minimum of Integer list	Integer → Real
AMIN1(X,Y,...)	Real minimum of Real list	Real → Real
MIN0(I,J,...)	Integer minimum of Integer list	Integer → Integer
MIN1(X,Y,...)	Integer minimum of Real list	Real → Integer
DMIN1(X,Y,...)	Double minimum of Double list	Double → Double

The transfer of sign functions return (sign of the second argument) * (absolute value of first argument)

SIGN(X,Y)	Real transfer of sign	Real → Real
ISIGN(I,J)	Integer transfer of sign	Integer → Integer
DSIGN(X,Y)	Double precision transfer of sign	Double → Double

Positive difference functions return the first argument minus the minimum of the two arguments

DIM(X,Y)	Real positive difference	Real → Real
IDIM(I,J)	Integer positive difference	Integer → Integer

Exponential functions return the value of e raised to the argument power

EXP(X)	e^x	Real → Real
DEXP(X)	e^x	Double → Double
CEXP(Z)	e^z	Complex → Complex
ALOG(X)	Returns $\log_e(X)$	Real → Real
ALOG10(X)	Returns $\log_{10}(X)$	Real → Real
DLOG(X)	Returns $\log_e(X)$	Double → Double
DLOG10(X)	Returns $\log_{10}(X)$	Double → Double
CLOG(Z)	Returns $\log_e$ of complex argument	Complex → Complex
SQRT(X)	Square root of Real argument	Real → Real
DSQRT(X)	Square root of Double precision argument	Double → Double
CSQRT(Z)	Square root of Complex argument	Complex → Complex
SIN(X)	Real sine	Real → Real
DSIN(X)	Double precision sine	Double → Double
CSIN(Z)	Complex sine	Complex → Complex
COS(X)	Real cosine	Real → Real
DCOS(X)	Double precision cosine	Double → Double
CCOS(Z)	Complex cosine	Complex → Complex
TANH(X)	Hyperbolic tangent	Real → Real
ATAN(X)	Real arc tangent	Real → Real
DATAN(X)	Double precision arc tangent	Double → Double
ATAN2(X,Y)	Real arc tangent of (X/Y)	Real → Real
DATAN2(X,Y)	Double precision arc tangent of (X/Y)	Double → Double
CONJG(Z)	Complex conjugate, if Z=X+i*Y CONJG(Z)=X-i*Y	Complex → Complex
RAN(I,J)	Returns a random number of uniform distribution over the range 0 to 1. I and J must be integer variables and should be set initially to 0. Resetting I and J to 0 regenerates the random number sequence. Alternate starting values for I and J will generate different random number sequences.	Integer → Real

System Subroutines

CALL ASSIGN (lun,name,icnt,mode,control,numbuf)
Allows the association of device and/or filename information with a logical unit number

CALL DATE (array)
Returns the current date in 'array'

CALL EXIT
Equivalent to the STOP statement without message

CALL IDATE (mon,iday,iyear)
Returns three integer values representing the current month, day, and year

RAN (i_1,i_2)
A function call to the random number generator

CALL RANDU (i_1,i_2,x)
A subroutine call to the random number generator

CALL SETERR (number,ncount)
Allows user to specify the disposition of OTS detected errors

CALL USEREX (name)
Pass control to routine 'name' as part of program termination

Logical Device Assignments

Logical Unit Number	Default Device
1	SY:
2	DK:
3	DK:
4	DK:
5	TT: (input)
6	LP:
7	TT: (output)
8	PR:
9	PP:

Switches

/A Include compilation statistics

/D Compile lines with D in column one

/E Read a full 80 columns of each record

/H Print a list of compiler switches

/L :n Specify listing options

n is any one or sum of the following
- 0 list diagnostics only
- 1 list source program only
- 2 list storage map
- 4 list generated code

/N:m Specify maximum number of logical units that may be concurrently open (m=octal constant between 1 and 17)

/O Include options-in-effect

/P Disable the global optimizer

/R:m Specify maximum record size allowed (m=octal constant between 4 and 7777)

/S Suppress Internal Statement Numbers

/T Allocate two words for default length of integer variables

/U Disable USR swapping at run-time

/V Disable vectoring of arrays

/W Enable compiler warning diagnostics

Compiler Error Diagnostics

B Columns 1-5 of a continuation line are not blank; columns ignored and compilation continues

C Illegal continuation; continued line ignored and compilation continues

E Missing END sentinel; an END sentinel is provided

H Hollerith string or quoted literal string too long; statement aborted

I Non-FORTRAN character used; character ignored and compilation continues

K Illegal statement label definition; illegal statement label ignored and compilation continues

L Line too long; line truncated to 80 characters and compilation continues

M Multiply defined label; label ignored

P Statement contains unbalanced parentheses; statement aborted

S Syntax error; statement aborted

U Statement not legal FORTRAN statement; statement aborted

Fatal Compiler Error Diagnostics

C Too many constant subscripts employed in statement; simplify statement

O Unrecoverable error occurred while Compiler was writing object file (.OBJ); rectify hardware problem or allow more space for output

P Optimizer push down overflow; simplify complex statements

R Unrecoverable hardware error occurred while Compiler was reading source file; rectify hardware problem

S Subexpression stack overflow; simplify complex statements

T Memory Overflow; break up program into subprograms or compile on larger machine

W Unrecoverable error occurred while Compiler was writing listing file; rectify hardware problem or allow more space for listing file

Y Code generation stack overflow; simplify complex statements

Z Compiler error; report to DIGITAL

Compiler-Secondary Phase-Errors

These error messages are self-explanatory when printed and are not included here. Refer to Appendix G of the *RT-11 System Reference Manual* for detailed definitions.

OTS Error Diagnostics

0	FATAL	NON-FORTRAN ERROR CALL
1	FATAL	INTEGER OVERFLOW
2	FATAL	INTEGER ZERO DIVIDE
3	FATAL	COMPILER GENERATED ERROR
4	WARNING	COMPUTED GO TO OUT OF RANGE
5	COUNT:3	INPUT CONVERSION ERROR
6	IGNORE	OUTPUT CONVERSION ERROR
10	COUNT:3	FLOATING OVERFLOW
11	IGNORE	FLOATING UNDERFLOW
12	FATAL	FLOATING ZERO DIVIDE
13	COUNT:3	SQRT OF NEGATIVE NUMBER
14	FATAL	UNDEFINED EXPONENTIATION OPERATION
15	FATAL	LOG OF NEGATIVE NUMBER
16	FATAL	WRONG NUMBER OF ARGUMENTS
20	FATAL	INVALID CHANNEL NUMBER
21	FATAL	NO AVAILABLE CHANNELS
22	FATAL	INPUT RECORD TOO LONG
23	FATAL	HARDWARE I/O ERROR
24	FATAL	ATTEMPT TO READ/WRITE PAST END OF FILE
25	FATAL	ATTEMPT TO READ AFTER WRITE
26	FATAL	RECURSIVE I/O NOT ALLOWED
27	FATAL	ATTEMPT TO USE DEVICE NOT IN SYSTEM
28	FATAL	OPEN FAILED FOR FILE
29	FATAL	NO ROOM FOR DEVICE HANDLER
30	FATAL	NO ROOM FOR BUFFERS
31	FATAL	NO AVAILABLE RT-11 CHANNEL
32	FATAL	FMTD–UNFMTD–RANDOM I/O TO SAME FILE
33	FATAL	ATTEMPT TO READ PAST END OF RECORD
34	FATAL	UNFMTD I/O TO TTY OR LPT
35	FATAL	ATTEMPT TO OUTPUT TO READ ONLY FILE
36	FATAL	BAD FILE SPECIFICATION STRING
37	FATAL	RANDOM ACCESS READ/WRITE BEFORE DEFINE FILE
38	FATAL	RANDOM I/O NOT ALLOWED ON TTY OR LPT
39	FATAL	RECORD LARGER THAN RECORD SIZE IN DEFINE FILE
40	FATAL	REQUEST FOR A BLOCK LARGER THAN 65535

41	FATAL	DEFINE FILE ATTEMPTED ON AN OPEN UNIT
42	FATAL	MEMORY OVERFLOW COMPILING OBJECT TIME FORMAT
43	FATAL	SYNTAX ERROR IN OBJECT TIME FORMAT
44	FATAL	2ND RECORD REQUEST IN ENCODE/ DECODE
45	FATAL	INCOMPATIBLE VARIABLE AND FORMAT TYPES
46	FATAL	INFINITE FORMAT LOOP
47	FATAL	ATTEMPT TO STORE OUTSIDE PARTITION
59	WARNING	USR NOT LOCKED
60	FATAL	STACK OVERFLOWED
61	FATAL	ILLEGAL MEMORY REFERENCE
62	FATAL	FORTRAN START FAIL
63	FATAL	ILLEGAL INSTRUCTION

Appendix D : BASIC op de PDP11

D.1. BASIC

BASIC is een conversationele programmeertaal die gebruik maakt van eenvoudige ENGELS-achtige statements en wiskundige notaties om bewerkingen te verrichten.
De verwerking van BASIC is interpretatief, hetgeen wil zeggen dat een statement pas op het moment van executie vertaald wordt. Hierdoor is de verwerkingssnelheid van BASIC laag t.o.v. assembler en FORTRAN programma's. BASIC is uitermate geschikt voor kleine rekenprogramma's.
BASIC kan in twee verwerkingsmodes werken te weten:

1e Immediate mode - Een statement wordt direkt na het intypen geëxecuteerd.

2e Programmamode - Een serie statements wordt na een RUN commando geëxecuteerd.

In Immediate mode moet een statement zonder statement nummer getypt worden, voorbeelden:

```
A = 0.5
PRINT A        (Print commando wordt direkt uitgevoerd)
```

In Programma mode worden de statements van nummers voorzien:

```
10 A = 0,5
11 PRINT A
```

D.2. Algemene gegevens van BASIC.

D.2.1. -Getallen.

Algemeen . Real getallen in twee woords floating point.

. Bij uitvoer: (teken) x.xxxxx E (teken) n
mantisse exponent
in 6 decimalen.

. Integers: Getallen die als integers ingevoerd worden, blijven integer tot een bewerking omzetting naar floating point notatie nodig maakt.

D.2.2. - Variabelen

Variabelen worden aangegeven door 1 letter al of niet gevolgd door 1 cijfer.
Voorbeelden: A, B1, C9

Opmerking: Alle variabelen worden voor executie van het programma gelijk nul gemaakt.

- Arrays

Een array mag maximaal twee dimensionaal zijn.
Voorbeelden:

A(I)	1 dim. array
B(I,J)	2 dim. array

I en J integer expressies, met I,J ≠ 0.

D.2.3. - Operatoren

D.2.3.1. Rekenkundige operatoren

+	plus	A+B
-	min	B-A
*	maal	C*D
/	gedeeld door	D/C
↑	macht	P↑Q

Bewerkingsvolgorde: 1e Haakjes ( )
2e Teken (enkel minteken)
3e Machtsverheffen (van links naar rechts)
4e Vermenigvuldigen en delen(")
5e Optellen en aftrekken (")

D.2.3.1. Logische operatoren

Logische operatoren zijn te gebruiken in IF statements

=	is gelijk	IF A=B THEN.....
<	kleiner dan	
<=	kleiner of gelijk	
>	groter dan	
>	groter of gelijk	
><	ongelijk	

D.2.4. Programma's.

Een programma wordt opgebouwd uit regels met BASIC instrukties. Elke regel begint met een regelnummer. (1<regelnummer<65532). Om later regels te kunnen tussenvoegen is het verstandig in eerste instantie de regelnummers met b.v. 10 op te hogen. Programma regels worden in volgorde van oplopende regelnummers geëxecuteerd. Meerdere statements kunnen in een regel worden geplaatst. De regels moeten gescheiden worden met een \ karakter (Back slash - shift L).

Voorbeeld: 20 LET X=10\PRINTX\GOTO 10

D.3. Starten van en werken met BASIC.

D.3.1. Starten van BASIC.

Het BASIC programma werkt onder het RT11 operating systeem en kan met een RUN commando gestart worden. Voorwaarde is, dat het RT11 operating systeem gestart is.

Dit gaat alsvolgt:

Computer AAN
HALT/ENABLE schakelaar op HALT
Plaats 173000 in het Switch Register
HALT/ENABLE schakelaar op ENABLE
Druk START
Hierna meldt RT11 zich met:
RT11 SJ.....
.
Na de punt moet, om BASIC te starten getypt worden:
. R BASIC'CR' 'CR' =RETURN of CR toets

Hierna meldt BASIC zich met:
BASIC V.....

Er moet nu een van de volgende karakters getypt worden:
A of 'CR' : alle functies worden aan BASIC gekoppeld
N : geen functies worden aan BASIC gekoppeld
I : per functie vraagt BASIC of gekoppeld moet worden.

Hierna typt BASIC:
READY
en is klaar om gebruikerscommando's of programma's uit te voeren.

D.4. BASIC voorbeeld:

```
  1. REM   SORTEERPROGRAMMA
  2  PRINT "TYPE TELKENS NA EEN ? EEN WILLEKEURIG GETAL"
  3  PRINT "NA 10 GETALLEN WORDEN DEZE IN GESORTEERDE"
  4  PRINT "VOLGORDE OP DE TERMINAL GEPRINT"
  5  DIM  A(10)
 10  FOR  B=1 TO 10
 11  INPUT A(B)
 12  NEXT B
 20  FOR I=1 TO 9
 30  FOR J= (I+1) TO 10
 35  IF  A(I)<A(J) THEN 50
 40  C=A(I)
 41  A(I)=A(J)
 42  A(J)=C
 50  NEXT J
 60  NEXT I
100  FOR I = 1 TO 10
110  PRINT A(I)
120  NEXT I
130  STOP
140  END
```

D.5.Statements - Functies - Commando's - foutmeldingen

Op de volgende pagina's is een overzicht gegeven van de BASIC statements, functies, commando's en foutmeldingen

Overzicht van BASIC RT11.

BASIC/RT11

Statements

CALL "function name" [(argument list)]
Calls assembly language user functions from BASIC program

CHAIN "file descriptor" [LINE number]
Terminates execution of user program, loads and executes specified program starting at line number, if included

CLOSE [{VFn / #n}]
Closes logical file specified; if no file specified, closes all open files

DATA data list
Used in conjunction with READ to input data into executing program

DEF FN letter (argument)=expression
Defines user function to be used in program

DIM variable(n), variable(n,m), variable$(n), variable$(n,m)
Reserves space for lists and tables according to subscripts specified after variable name

END
Placed at physical end of program to terminate program execution

FOR variable = expr1 TO expr2 [STEP expr3]
Sets up loop to be executed specified number of times (expr = expression)

GOSUB line number
Transfers control to specified line of subroutine

GO TO line number
Unconditionally transfers control to specified line in program

IF expression rel.op expression {THEN / GO TO} line number
Conditionally transfers control to specified line of program

IF END #n {THEN / GO TO} line number
Tests for end file on sequential input file #n

INPUT var list
Inputs data from terminal to specified variables

INPUT #expression: var list
Inputs data from particular input device, as specified in OPEN statement, to specified variables

[LET] variable=expression
Used to assign value to specified variable

[LET] VFn(i)=expression
Used to set value of virtual memory file element

NEXT variable
Placed at end of FOR loop to return control to FOR statement

OPEN

OPEN file [FOR {INPUT / OUTPUT}] [(b)] AS FILE #n [DOUBLE BUF]
Opens sequential file for input or output as specified

OPEN file [FOR {INPUT / OUTPUT}] [(b)] AS FILE VFnx (dimension)=string length
Opens virtual memory file for input or output

OVERLAY "file descriptor"
Overlays or merges program currently in memory with specified file, and continues execution

PRINT expression list
Outputs data to terminal

Statements (continued)

PRINT #expression: expression list
Outputs to particular output device, as specified in OPEN statement

RANDOMIZE
Causes random number generator to calculate different random numbers every time program is run

READ variable list
Assigns values listed in DATA statement to specified variables

REM comment
Inserts explanatory comments into BASIC program

RESTORE
Resets data block pointer so same data can be used again

RESTORE #n
Rewinds input sequential file #n to beginning

RETURN
Returns program control to statement following last GOSUB statement

STOP
Used at the logical end of the program to terminate execution

Key Commands

ALTMODE (ESC)
Deletes entire current line

CTRL C
Interrupts execution of command or program and returns control to RT-11 Monitor

CTRL O
Stops output to terminal and returns BASIC to READY message when program or command execution is completed

CTRL U
Deletes entire current line

(SHIFT/O)
Deletes last character typed

RUBOUT
Deletes last character typed

Math Functions

ABS(x)
Returns absolute value of x

ATN(x)
Returns arctangent of x as angle in radians in range $+\pi/2$ or $-\pi/2$

BIN(x$)
Computes integer value of string of 1's and 0's

COS(x)
Returns cosine of x radians

EXP(x)
Returns value of e† x where e=2.71828

INT(x)
Returns greatest integer less than or equal to x

LOG(x)
Returns natural logarithm of x

OCT(x$)
Computes integer value from string of blanks and digits from 0 to 7

RND(x)
Returns random number between 0 and 1

SGN(x)
Returns value indicating sign of x

SIN(x)
Returns sine of x radians

SQR(x)
Returns square root of x

TAB(x)
Causes terminal type head to tab to column number x

Edit and Control Commands

CLEAR
Sets array and string buffers to nulls and zeroes

LIST
Lists user program currently in memory on terminal

LIST line number

LIST -line number

LIST line number- [END]

LIST line number-line number
Lists specified program line(s) on terminal

LISTNH
Lists user program currently in memory on terminal (no header)

LISTNH line number

LISTNH -line number

LISTNH line number- [END]

LISTNH line number-line number
Lists lines associated with specified numbers (no header)

NEW "file descriptor"
Does SCRatch and sets current program name to one specified

OLD "file descriptor"
Does a SCRatch and inputs program from specified file

RENAME "file descriptor"
Changes current program name to one specified

REPLACE "file descriptor"
Replaces specified file with current program

RUN
Executes program in memory

RUNNH
Executes program in memory (no header)

SAVE "file descriptor"
Outputs program in memory as specified file

SCRatch
Erases entire storage area

String Functions

ASC(x$)
Returns as decimal number 7-bit internal code for 1-character string (x$)

CHR$(x)
Generates 1-character string having ASCII value of x

DAT$
Returns current date in format 07-May-73

LEN(x$)
Returns number of characters in string (x$)

POS(x$,y$,z)
Searches for and returns position of first occurrence of y$ in x$ starting with z^{th} position

SEG$(x$,y,z)
Returns string of characters in positions y through z in x$

STR$(x)
Returns string which represents numeric value of x

TRM$(x$)
Returns x$ without trailing blanks

VAL(x$)
Returns number represented by string (x$)

Error Messages

?ARG AT LINE xxxxx	ARGUMENT ERROR
?ATL AT LINE xxxxx	ARRAYS TOO LARGE
?BDR AT LINE xxxxx	BAD DATA READ
?BRT	BAD DATA-RETYPE FROM ERROR
?BSO AT LINE xxxxx	BUFFER STORAGE OVERFLOW
?DCE AT LINE xxxxx	DEVICE CHANNEL ERROR
?DFO AT LINE xxxxx	DISPLAY FILE OVERFLOW
?DNR	DEVICE NOT READY
?DV0 AT LINE xxxxx	DIVISION BY 0
?ETC AT LINE xxxxx	EXPRESSION TOO COMPLEX
?FDE AT LINE xxxxx	FILE DATA ERROR
?FIO AT LINE xxxxx	FILE I/O ERROR
?FNF AT LINE xxxxx	FILE NOT FOUND
?FNO AT LINE xxxxx	FILE NOT OPEN
?FTS AT LINE xxxxx	FILE TOO SHORT
?FWN AT LINE xxxxx	FOR WITHOUT NEXT
?GND AT LINE xxxxx	GOSUBS NESTED TOO DEEPLY
?IDF AT LINE xxxxx	ILLEGAL DEF
?IDM AT LINE xxxxx	ILLEGAL DIM
?ILN	ILLEGAL NOW
?ILR AT LINE xxxxx	ILLEGAL READ
?INS AT LINE xxxxx	IMPROPERLY NESTED SUBPICTURES
?LTL	LINE TOO LONG
?NBF AT LINE xxxxx	NEXT BEFORE FOR
?NER AT LINE xxxxx	NOT ENOUGH ROOM
?NPR	NO PROGRAM
?NSM AT LINE xxxxx	NUMBERS AND STRINGS MIXED
?OOD AT LINE xxxxx	OUT OF DATA
?OVF AT LINE xxxxx	OVERFLOW
?PTB	PROGRAM TOO BIG
?PWF AT LINE xxxxx	POWER FAIL
?RBG AT LINE xxxxx	RETURN BEFORE GOSUB
?RPL	FILE SAVED ALREADY EXISTS
?SOB AT LINE xxxxx	SUBSCRIPT OUT OF BOUNDS
?SSO AT LINE xxxxx	STRING STORAGE OVERFLOW
?STL AT LINE xxxxx	STRING TOO LONG
?SYN AT LINE xxxxx	SYNTAX ERROR
?TLT	LINE TOO LONG TO TRANSLATE
?UFN AT LINE xxxxx	UNDEFINED FUNCTION
?ULN AT LINE xxxxx	UNDEFINED LINE NUMBER
?WLO AT LINE xxxxx	WRITE LOCKOUT
?↑ER AT LINE xxxxx	↑ ERROR